Dynamic Voltage Scaling
on MPEG Decoding

Donghwan Son
School of Engineering
Information and Communications University
2001
Dynamic Voltage Scaling on MPEG Decoding
Dynamic Voltage Scaling
on MPEG Decoding

Advisor : Professor Chansu Yu

by

Donghwan Son
School of Engineering
Information and Communications University

A thesis submitted to the faculty of Information and Communications University in partial fulfillment of the requirements for the degree of Master of Science in the School of Engineering

Daejeon, Korea

Approved by

____________________
Professor Chansu Yu
Major Advisor
Dynamic Voltage Scaling

on MPEG Decoding

Donghwan Son

We certify that this work has passed the scholastic standards requested by the Information and Communications University as a thesis for the degree of Master


 Approved :

Chairman of the Committee
Chansu Yu, Associate Professor
School of Engineering

Committee Member
Ben Lee, Associate Professor
Dep. of Electrical and Computer Engineering
Oregon State University

Committee Member
Dong Soo Han, Assistant Professor
School of Engineering
Abstract

Mobile devices require long battery lifetime while still delivering high performance. 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). Previous DVS algorithms usually rely on workload history. But it may not be efficient when the workload variability is high as in MPEG decoding, which will become one of the most important applications in mobile computing. This thesis presents two DVS algorithms on MPEG decoding. One is DVS with delay and drop rate minimizing algorithm (DVS-DM) where voltage is determined by previous workload. Another algorithm predicts MPEG decoding time using the sizes of next frames, and scales voltage according to the predicted MPEG decoding time and previous workload (DVS with predicted decoding time or DVS-PD). We compare four MPEG decoders: the first one is original MPEG decoder from Boston University, the second one is MPEG decoder with shutdown mechanism, the third implements DVS-DM algorithm, and the last one runs DVS-PD. Simulation results show that DVS-PD saves more energy than other algorithms due to more accurate
prediction of future workload. We also found that the amount of energy saving with DVS-PD is related with error rate of the predictor, which implies that if decoding time is predicted more accurately, DVS algorithm can be more efficient.
## Contents

Abstract ......................................................................................... i

Contents ....................................................................................... iii

List of Figures ................................................................. v

List of Tables ................................................................. vi

I. Introduction ........................................................................... 1

II. Dynamic Voltage Scaling .............................................. 4

  2.1 Dynamic Power Management ....................................... 4

  2.2 Dynamic Voltage Scaling ........................................... 9

  2.3 DVS Algorithms ......................................................... 12

III. MPEG Decoding ........................................................... 19

  3.1 MPEG Decoding ....................................................... 19

  3.2 Predicting MPEG Execution Time ............................... 21

IV. Proposed Algorithms ................................................... 22

  4.1 DVS with Delay and Drop Rate Minimizing Algorithm (DVS-DM) ................................................................. 22

  4.2 DVS with Predicted MPEG Decoding Time (DVS-PD) ......... 23
V. Experimental Environments ........................................ 27
  5.1 MPEG Decoder .................................................. 27
  5.2 Sample Streams................................................... 27
  5.3 Assumptions....................................................... 34

VI. Performance Evaluation ......................................... 36
  6.1 Decoder with Shutdown Mechanism............................. 38
  6.2 Decoder with DVS-DM........................................... 39
  6.3 Decoder with DVS-PD........................................... 39

VII. Conclusions....................................................... 42

국문요약 ............................................................. 44

References ............................................................ 46

Acknowledgement ...................................................... 51

Curriculum Vitae ..................................................... 52
List of Figures

Figure 1 Voltage Scheduling Graphs by Two Mechanism .................. 11
Figure 2 Power Reduction Using DVS................................................. 12
Figure 3 Optimal Coincident Task Scheduling ............................... 15
Figure 4 Greedy Algorithm Under-Estimation ................................. 16
Figure 5 Thread-based Voltage Scheduling ..................................... 18
Figure 6 Example of MPEG Frame Sequence ................................. 20
Figure 7 Scene Fluctuations ............................................................ 29
Figure 8 Average GOP Decoding Times .......................................... 29
Figure 9 GOP Decoding Time Deviations ....................................... 29
Figure 10 Frame Size Sequences of Sample Streams ......................... 30
Figure 11 Sequences of GOP Decoding Time .................................. 32
Figure 12 Energy Consumptions .................................................... 37
Figure 13 Played Frame Ratio ........................................................ 38
Figure 14 Error Rates of Sample Streams ...................................... 40
List of Tables

Table 1: Frames Per Second and Typical Applications………………… 19
Table 2: Sample Video Clips .................................................. 28
I. Introduction

Efficiency in energy consumption has become an important aspect of modern computing. This is particularly true for portable systems, such as personal computing and communication devices, which have limited battery capacities.

Power consumption of a processor is the most critical factor as the mobile devices should execute more and more complex works, and this demands faster processor speed which results in increased power consumption. Moreover, in ubiquitous computing devices and microsensor networks [1], energy efficiency of a processor is crucial 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 level, which exploits the quadratic dependence of power on voltage. Reducing the supply voltage however increases circuit delay and decreases clock speed. It is ineffective as real systems have latency critical tasks. The solution for this problem 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 supply voltage that can be varied at run time according to application time constraints [2, 3]. 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 [4, 5].

Dynamic voltage scaling (DVS) allows a processor to dynamically change its speed and voltage at run time, increasing energy efficiency. Taking advantages of DVS requires algorithms, termed voltage schedulers, to determine the operating speed of the processor at run time. In [6-8], interval based DVSs have been proposed, which periodically analyze system utilization.
at a global level. It is simple since no direct knowledge of individual threads or programs is needed. Voltage of the next interval is determined by the workload of the previous intervals. Interval based scheduling is easy to implement, but it often incorrectly predicts future workloads and hurts the quality of service. For realtime systems, there proposed several DVS algorithms which minimize energy consumption while all tasks complete on or before deadlines [9-14].

On the other hand, the mobile computing environment has been changed rapidly in recent years. One of the variation is the growing number of multimedia applications, ranging from video games and movie players to sophisticated virtual reality environment on mobile devices. MPEG decoder is one of those applications and widely used in mobile devices from notebook computers to communication devices.

The core of MPEG decoder is an inverse discrete cosine transform (IDCT) which is used to decompress 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, the time to process any given frame is constant according to the frame rate of the stream. The variability of this workload and the realtime constraint causes the interval based DVS algorithm to fail: predicting the next workload based on previous workload is difficult and a wrong prediction causes frames to be dropped.

In this thesis, 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 proposed 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 as suggested in [15], and we found that it is reasonably accurate than that based on previous workload.

The rest of the thesis is organized in the following way. Chapter 2 presents the background on DVS and DVS algorithms. Chapter 3 provides the related work on MPEG and MPEG decoding. In Chapters 4 and 5, two DVS algorithms are proposed and experimental environments are presented, respectively. Chapter 6 evaluates the performance of proposed algorithms. The thesis is concluded in Chapter 7.
II. Dynamic Voltage Scaling

Before presenting our approach, we first provide a general overview of dynamic power management and dynamic voltage scaling. Section 2.1 introduces power management technologies in general, and Section 2.2 discusses the dynamic voltage scaling at the digital circuit level and how it contributes to power reduction. Section 2.3 surveys several DVS algorithms proposed in the literature.

2.1 Dynamic Power Management

In this section, we analyze techniques for controlling the power state of a system and its components. Components can be any kind of devices that consumes energy.

There are three aspects of power management [16]. The first one is transition strategy which determines when a component should switch from one mode to another. This is trivial if transitions between power states are instantaneous: negligible power and performance costs are paid for performing state transitions. In such a system, optimum policy is greedy: as soon as the system is idle, it can be transitioned to the deepest sleep-state available. On the arrival of a request, the system is instantaneously activated.

Unfortunately, most components have non-negligible power and performance costs for power state transition. For instance, if entering a low power state requires processor shutdown, returning from this state to active state requires a time for turning on and stabilizing the processor. This latency causes power consumption and performance reduction. Therefore more
sophisticated policies are needed to efficiently transit between states considering power saving and performance tradeoff.

To optimize power policy under performance constraints, the information of future workload is important as the achievable power savings depend on the workload. If we can predict future workload exactly, which is not usually possible, optimal power saving can be obtained. The policies to predict future workload can be divided by two strategies: predictive technology and stochastic control, which are surveyed in following subsections.

The second aspect of the power management is the load change strategies. Load change strategy modifies the load on a component to increase the use of low-power modes. One example is disk caching which reduces the disk access so as to increase the spin-down state of a disk.

Adaptation is the third aspect, which considers how software can be modified to permit novel power-saving of a component. For this strategy, detailed information of the component, and whole system in which the component is involved, is needed. For example, in evaluating a power management strategy, we should consider the component-specific information such as usage pattern of battery capacity by that component, power bottlenecks in the system and effects on other components.

2.1.1 Predictive Techniques

The rationale in all predictive techniques is that of exploiting the correlation between the past history of the workload and its near future in order to make reliable predictions about future events. Several predictive techniques are presented below.
Fixed Timeout

*Fixed timeout* uses elapsed idle time to predict the total duration of current idle time. When an idle period begins, a timer is started with duration $T$. If the system is still idle after $T$, then the power management forces the transition to shutdown state. The system remains off until it receives a request from the environment that signals the end of the idle period.

Timeouts have two advantages: they are general and their safety of prediction can be improved simply by increasing the timeout values. The weak point of this strategy is long timeouts cause a large number of under-predictions, a predicted idle period longer than the actual one, that represent missed opportunity of saving power, and a sizeable amount of power is wasted for the timeout to expire. Another limitation of timeout schemes is that they always pay a performance penalty upon wakeup.

Predictive shutdown

Two *predictive shut-down schemes* have been proposed by Srivastava et al. [17]. Let $T_{on}[i]$ and $T_{off}[i]$ be the time spent by the system in the $i$th visit to the on and the off state, respectively. Further, let $T_{cost}$ be the time overhead associated with the process of shutting down. A heuristic rule is used to predict whether $T_{off}[i] \geq T_{cost}$ and if so it is decided to shut the processor down.

The first scheme, is to use regression analysis to arrive at a model for predicting $T_{off}[i]$ as following equation.

$$T_{off}[i] = 0.0740018 + 0.553733 T_{off}[i-1] - 0.00947348 T_{off}[i-1]^2$$
$$+ 1.42233 T_{on}[i] + 1.13883 T_{off}[i-1] T_{on}[i] - 1.49143 T_{on}[i]^2.$$  

The format of the non-linear regression is decided heuristically, while the fitting coefficients can be computed with standard techniques. The main
limitations of this approach are: (i) there is no automatic way to decide the type of regression equation; (ii) off-line data collection and analysis are required to construct and fit the regression model.

The second approach is based on that $T_{on}[i]$ corresponds to the most recent history of $T_{off}[i]$. This suggests that a large value of $T_{on}[i]$ is followed by a small $T_{off}[i]$ with a very high probability. Based on this observation, they derived a simple filtering (or thresholding) rule as shown below, which filters out the idle period fulfilling the above condition and the system enters the sleep state otherwise.

$$T_{off}[i] \geq T_{cost} \leftrightarrow T_{on}[i] \leq T_{on\_threshold}$$

Hwang et al. [18] extended the above predictive shut-down scheme using exponential-average approach to predict the idle period. This approach depends solely on the history of the previous idle periods, thus no off-line data collection and analysis are required. The recursive prediction formula is shown below.

$$t_{off}[i] = \alpha T_{off}[i - 1] + (1 - \alpha) t_{off}[i - 1]$$

where $t_{off}[i]$ is the $i$th predicted idle time and $\alpha$ is a constant attenuation factor in the range between 0 and 1.

2.1.2 Stochastic Control

A common feature of all predictive shutdown is that policies are formulated heuristically, then tested with simulations or measurements to assess their effectiveness. Another commonality is that the system model for policy optimization is very abstract, and abstraction introduces uncertainty. Hence, it may be safer, and more general, to assume a stochastic model for the system.
The policy optimization technique proposed by Paleologo et al. [19] solves the policy optimization problem with globally optimal results using an approach based on discrete-time Markov decision processes (DTMDP) [20]. The solution is computed in polynomial time by solving a linear optimization problem.

In the model of [19], time is divided into small intervals of length \( L \). It is assumed that the system can only change its state at the beginning of a time interval. During interval \((jL, (j+1)L)\), the transition probability of the system depends only on the state of the system at time \( jL \) (hence, the Markovian property) and the command issued by the power manager. The system model consists of four components: a power manager (PM), a service provider (SP), a service requestor (SR) and a service request queue (SQ). Once the model and its parameters have been determined, an optimal power management policy is obtained to achieve the best power-delay trade-off.

The DTMDP approach requires that all state transitions follow stationary geometric distribution, which is not true in many practical cases. Another shortcoming is that the model does not distinguish between the busy state and the idle state of the SP, therefore the state transition probability of the system model cannot be calculated accurately. These shortcomings are overcome in [21] using continuous-time Markov decision process (CTMDP). In CTMDP model, PM issues commands upon event occurrences instead of at discrete time settings; system transitions are assumed to follow exponential distribution; the correlation between the state of the SQ and the state of the SP is considered; a transfer state is induced in the model of SQ to distinguish between the busy and idle state of the SP. CTMDP model was further generalized with semi-Markov decision process model (SMDP) [22, 23]. SMDP model can treat a general distribution occurring at the same time with
an exponential distribution.

Although exponential distribution model can be used to model arrivals in the active states, the arrivals in the idle states are better modeled with Pareto distribution after filtering out small interarrival times as shown in [24]. This is because models that are based on exponential arrival times in the idle state can have high energy costs and large performance penalty because the power manager makes one decision as soon as the system goes idle. If the decision is to stay awake, the system will wait until another arrival before revising the decision, possibly missing large idle times, such as lunch break. By this observation, \textit{time-indexed Markov chain SMDP model} is presented in [24]. This model combines the advantages of event-driven SMDP model with DTMDP model. Policy decisions are made in event-driven manner in all states but idle state thus saving power by not forcing policy re-evaluations. In the idle state, policy decision are re-evaluated until the transition is made into the sleep state, thus saving power during longer breaks.

\subsection{2.2 Dynamic Voltage Scaling}

In digital static CMOS circuit, energy consumed by each operation is given by the following equation [7].

\[ E \propto C_{eff}V^2 \]

where \( C_{eff} \) is the effective switching capacitance of that operation and \( V \) is the supply voltage.

At the same time, delay of the circuit is given by:

\[ delay \propto \frac{V}{(V-V_h)^\alpha} \]
where $\alpha$ ranges from 1 to 2, and $V_k$ depends on threshold voltage and sometimes voltage at which velocity saturation occurs [7]. Regardless the exact value of $V_k$, the combination of the two equations always implies that lowering the supply voltage will result in longer delay, but at the same time, quadratic decrease in energy. This contrasts to other power saving schemes where the switching frequency is changed instead of supply voltage. If only the switching frequency were changed, instantaneous energy consumption is altered, but the total energy consumed for each operation remains the same. On the other hand, if supply voltage is changed, and we are still running at the maximum allowable frequency, energy consumption for the entire operation is reduced.

*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 [6, 7]. As the workload is usually not constant but bursty, there are idle times in which CPU does not work [25]. One of conventional way of reducing CPU power consumption is *shutdown mechanism* which stops CPU operation while idle. But as the energy is the quadratic function of voltage, power consumption on DVS technique is more effective than the shutdown mechanism.

Figure 1 shows an example of voltage scheduling graph. The X-axis represents time and the Y-axis represents voltage or processor speed. In this graph, the voltage and frequency are assumed to be exactly proportional. In the figure, the computation required by a task is proportional to the ‘area under the curve’ for that task. Voltage and processor speed is normalized to the range $[0…1]$. The height of a task (processor speed or voltage) and work performed determine its energy consumed.

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, then the energy consumed for three jobs is 8 energy units. In case of
dynamic voltage scaling, voltage can be reduced to 3/5, 1/3 and 2/3 for three
jobs, respectively, which results in energy consumption of only 2.97 energy
units \((3/5)^3 \times 5 + (1/3)^3 \times 3 + (2/3)^3 \times 6 = 2.97\) as shown in Figure 1.

Figure 2 shows a plot of power vs. normalized workload [26]. The straight
line represents the power consumption with the shutdown mechanism. Power
is reduced in a linear fashion since the energy per operation is fixed, and only
the number of operations changes. The lower curve represents the case for
dynamic voltage scaling. If the workload for a given sample period is less than
the peak, then the delay of the processing element can be increased by a factor
of 1/workload without loss in throughput, allowing the processor to operate at
a lower supply voltage.

Figure 1: Voltage Scheduling Graphs by Two Mechanism
Current VLSI technology already supports a set of 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 [7], 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 [27].

![Power Reduction Using DVS](image)

Figure 2: Power Reduction Using DVS [26]

### 2.3 DVS Algorithms

#### 2.3.1 DVS based on Previous Workload for Non-Realtime System

As the dynamic voltage scaling relies on the workload of system, information on workload is critical to adapt DVS efficiently. But it is not usually possible
to know exact future workload of non-realtime system. Instead, there are several prediction algorithms to estimate workload of the near future heuristically. Most of the algorithms divide time by uniform-length intervals and analyze workloads of previous intervals to estimate that of the next interval. The estimation is based on two new parameters as defined in [6]; \textit{run\_percent} and \textit{excess\_cycles}. \textit{run\_percent} is the fraction of cycles in an interval where the CPU is active. \textit{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 algorithms to predict workload based on the two parameters are presented here.

\textbf{PAST [6]}

PAST calculates how busy the last completed interval was (including excess cycles brought into that interval). It then predicts that the coming interval will be equally busy. PAST does surprisingly well in spite of its algorithmic simplicity. But in a bursty trace, the assumption that adjacent intervals will be similar is in fact almost certainly wrong, which causes room for improvement of this algorithm.

\textbf{FLAT [7]}

Weak on prediction, it simply tries to smooth speed to a global average. FLAT takes an input parameter which must be real number on range [0, 1]. It set’s new \textit{run\_percent} to be constant, and speed is set to be fast enough to complete the \textit{run\_percent} plus the \textit{excess\_cycles} being pushed into the coming interval. While FLAT wants to keep \textit{run\_percent} as flat as
possible, it also responds effectively to excess_cycles. Indeed, speed is always set fast enough to complete at least the excess_cycles, so no work may be delayed more than one interval.

**LONG_SHORT** [7]
LONG_SHORT is a more predictive policy, one which attempts to find a golden mean between local behavior and a more long-term average. It look up the last run_percent. The three most recent run_percent constitute the short-term past; the remaining nine, the long-term past. The prediction for the coming run_percent is then a weighted sum of these twelve values, where each short-term value is given a weight.

**AGED_AVERAGES** [7]
A cleaner variant of LONG_SHORT, AGED_AVERAGES employs an exponential-smoothing method, attempting to predict via a weighted average: one which geometrically reduces the weight given to each previous interval as we go back in time.

**CYCLE** [7]
The CYCLE policy was inspired by run_percent plots which look quite cyclical. This algorithm tries to find cycle by previous run_percent and set speed. If no good cycle is found, FLAT algorithm is used to set speed.

### 2.3.2 DVS in Real-time System
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: 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 3 depicts three tasks with $S_i = 0$ and differing deadlines. For such a schedule, with ordered deadlines $D_i \leq D_k \forall i < k$, the optimal voltage schedule can be found with the following algorithm, which runs in $O(n)$ time to schedule $n$ tasks. The intermediate workload, $W_i$, determines the optimal schedule for all tasks with deadlines at or before $D_i$. $P_i$ calculates the processor speed to use such that future deadlines are not violated.

- Let $W_i = \frac{1}{D_i} \sum_{n=i}^{i} C_n$
- Let $P_i = MAX(W_k, k \geq i)$
- For any time $T$, find the minimum $j$ such that $T \leq D_j$; schedule the processor at speed $P_j$.

![Figure 3: Optimal Coincident Task Scheduling](image-url)
Extending this algorithm to the general case where $S_i \neq S_k \quad \forall \ i, k$ is non-trivial. A simple modification of the above algorithm, which considers a task only if $S_i \leq T$ and re-evaluates at any time $T = S_i$, has the drawback of initially under-estimating the resources required, as in Figure 4. This greedy algorithm also has the potential to schedule tasks such that no longer meet their deadlines by initially running too slowly.

Ishihara and Yasuura [9] considered 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 [10], a dynamic voltage scheduling algorithm for realtime system is proposed. They defined an occupation period as the maximum value of period that the next executed task can use without violation of real-time constraints for the future tasks. That is, if the next task is assigned supply voltage so as to finish within the occupation period, the real-time constraints are always guaranteed. 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 the realtime system with two voltage modes (low mode and high mode), static and dynamic voltage scaling methods are proposed in [11]. In the static scheduling, each task is assigned an operational mode in an off-line computation by branch-and-bound approach. At the execution stage, the task dispatcher in realtime OS can initiate an operational mode change according to the assigned mode of the executing task. The mode assignment is formulated as a subset sum problem subject to time-demand schedulability analysis. Differing from the static approach, a dynamic scheduling approach can consider the existing workload imposed on the system. Instead of running a subset of tasks in high mode completely, by looking into the minimal amount of execution duration in low mode, if the duration is not zero, the processor can stay in low mode no matter which task is running. On the other hand, when there is no more low mode execution duration, then, the processor must be operated in high mode until either all waiting tasks are completed or an arriving task brings in additional low mode duration. the main advantage of dynamic approach against static approach is that high mode operation is executed as late as possible. If tasks arrive with variant phases, or the real execution times are less than worst case, the greedy approach of using low mode execution duration first can avoid unnecessary switches to high mode.

More recently, investigation has begun into thread-based voltage schedulers, which require knowledge of individual thread deadlines and computation required [12]. Given such information, thread-based schedulers can calculate the optimal speed and voltage setting, resulting in minimized energy consumption. A sample thread-based voltage scheduling is shown in Figure 5.
Figure 5: Thread-based Voltage Scheduling
III. MPEG Decoding

This chapter discusses the basic structure of MPEG stream as well as the decoding time and the decoding strategy for real time play.

3.1 MPEG Decoding [28]

The MPEG video compression standard defines a video stream as a sequence of still images (frames) that are displayed at some rate: this rate is referred as frames per second (fps). Standard MPEG provides eight values of fps as displayed with typical applications in Table 1. 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 [29-33].

<table>
<thead>
<tr>
<th>Frames Per Second</th>
<th>Typical Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>23.976</td>
<td>Movies on NTSC broadcast monitors</td>
</tr>
<tr>
<td>24</td>
<td>Movies, commercial clips, animation</td>
</tr>
<tr>
<td>25</td>
<td>PAL, SECAM, generic 625/50Hz component video</td>
</tr>
<tr>
<td>29.97</td>
<td>Broadcast rate NTSC</td>
</tr>
<tr>
<td>30</td>
<td>NTSC profession studio, 525/60Hz component video</td>
</tr>
<tr>
<td>50</td>
<td>Noninterlaced PAL/SECAM/625 video</td>
</tr>
<tr>
<td>59.94</td>
<td>Noninterlaced broadcast NTSC</td>
</tr>
<tr>
<td>60</td>
<td>Noninterlaced studio 525 NTSC rate</td>
</tr>
</tbody>
</table>

Table 1: Frames Per Second and Typical Applications [28]
From decoding time point of view, I frames usually take longer than other frames while B frames take the least. On the contrary, \textit{I-frame decoding time per byte (I\_DTPB)} is the shortest while \textit{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-frame and P-frame [15]. The frame decoding time can be predicted by frame type and size [15]. The decoding time of a frame is proportional to the size of the frame but differs by the type of the frame.

A \textit{Group of Pictures (GOP)} is a sequence of frames from one I frame to the next I frame. Most of encoders use a fixed \textit{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. Figure 6 shows an example of MPEG frame sequence which is composed of three GOPs, where the GOP pattern is IBBPBB.

![Figure 6: Example of MPEG Frame Sequence](image)

Another characteristic of a MPEG decoder is its realtime constraints [34]. The frame should be passed with a given frame per second either played or dropped. For this constraint, a decoder should monitor the current system speed with realtime clock, and decide each frame either to drop or play. Decoders usually maintain four realtime phases, such as \textit{DELAY-PHASE, B-PHASE, P-PHASE} and \textit{I-PHASE}. If the system is too fast to meet the realtime
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). This dropping algorithm is due to the fact that I frames and P frames are referred by P or B frames: dropping I or P frames cause other frames referring those to be dropped also.

3.2 Predicting MPEG Execution Times

MPEG decoding does not consume a constant amount of processing, 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 past behavior difficult.

In [15], the predictor for MPEG decoding is based on the frame type and the frame size with which the decoding time is shown to be related [15]. If the frame size is larger, then the decoding time is longer. With this predictor, they showed that they can predict the decoding time within 25% error. In the next chapter, we exploit this relationship to develop our proposed DVS algorithm.
IV. Proposed Algorithms

In this chapter, 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.

4.1 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-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 implement this algorithm is shown below.
static void scale_voltage_delay( )
{
    if((phase == I_PHASE) || (phase == P_PHASE))
        /* voltage is set to maximum voltage */
        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)
            return; /* keep current voltage */
        else /* scale down voltage */
            current_voltage = current_voltage -
                delay*decrease_factor;
        set_voltage(current_voltage);
    }
}

4.2 DVS with Predicted MPEG Decoding Time (DVS-PD)

The previous algorithm is one of the workload history based algorithms. 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 in the next Chapter.
We propose an efficient DVS algorithm which uses both previous history and estimated decoding time. 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 Chapter 3, when decoding a stream, each frame type has different decoding time per byte (DTPB). The DTPB for I, P and B-frame are not constant: these values differ by scene characteristics and vary throughout the MPEG decoding as the workload from other application changes. To adapt to workload changes and scene characteristics, our algorithm first maintain DTPBs by weighted sum of previous average of DTPBs and the last DTPBs in update_statistics routine as shown below. 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 of the next GOP are set according to the estimated GOP decoding time.

The balance_factor, which is determined by delay and drop rates as well as three parameters, I_P_balance, B_balance and DELAY_balance, decides the difference between the predicted workload and real workload of the recent interval. If the balance_factor is below zero, it means that the prediction 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, prediction is corrected by adding balance_factor to the estimated voltage, which
helps algorithm to be more adaptively cope with workload changes. We implemented this algorithm as below.

```java
static void scale_voltage_predict( )
{
    update_statistics( );
    get_GOP_size( );
    decode_time = estimate_decode_time( );
    set_voltage_predict( decode_time );
}

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 estimate_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_voltage_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 voltage 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;
    }
    voltage = std_voltage * d_time / std_dtime
        + balance_factor;
}
V. Experimental Environment

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

5.1 MPEG Decoder [35]

We modified MPEG decoder from Boston University. This decoder is based on Berkeley MPEG decoder and follows MPEG I standard. It added the functions of realtime play and multi-stream play on Berkeley MPEG decoder. To support the realtime play, the player dynamically adapts to drop frames to match the number of frames passed through the player to the frame rate given in the sequence header of the video stream.

5.2 Sample Streams

We selected six famous movies including two animations for evaluating our decoder with respect to energy consumption. For saving simulation time, instead of playing the whole movie, we used a portion (5000 frames from each movie) of 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 2, Figures 7, 8, 9 and 10.

The fluctuation of GOP decoding time, we are presenting here, is a degree of differences between adjacent GOP decoding times. High fluctuation means that the variation of GOP decoding time is high. Fluctuation is also related with the number of scenes. A scene is a segment of which the size of GOP are close
in value [36]. When a scene ends and another scene starts, there is large difference of GOP size between two scenes thus causes large fluctuation. The more scenes a stream is composed of, the larger the fluctuation. Another aspect that affects the fluctuation is the variation within a scene. Even in a scene, if the degree of activity of the objects in the scene changes much, the fluctuation becomes large. Fluctuation is obtained from the following equation. Action movies, such as Alien and Charlies Angel in our sample streams, usually have large fluctuations.

\[
Fluctuation = \sum_{i} \frac{(decode_{time_i} - decode_{time_{i-1}})^2}{num_{frames}}
\]

Deviation of GOP decoding time has similar value with fluctuation as shown in figure 9. From figures 10 and 11, which shows the sequences of frame sizes and GOP decoding time, respectively, from our sample streams, it is observed that if frame size varies highly, decoding time also varies highly.

<table>
<thead>
<tr>
<th>Movie</th>
<th>Average GOP Size (Byte)</th>
<th>Frames Per Second</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alien</td>
<td>96066</td>
<td>23.976</td>
</tr>
<tr>
<td>Conan</td>
<td>76778</td>
<td>30</td>
</tr>
<tr>
<td>The Close Shave</td>
<td>96065</td>
<td>23.976</td>
</tr>
<tr>
<td>X-men</td>
<td>76775</td>
<td>30</td>
</tr>
<tr>
<td>Hollow Man</td>
<td>96066</td>
<td>23.976</td>
</tr>
<tr>
<td>Charlie’s Angel</td>
<td>92131</td>
<td>25</td>
</tr>
</tbody>
</table>

Table 2: Sample Video Clips
Figure 7: Scene Fluctuations

Figure 8: Average GOP Decoding Times

Figure 9: GOP Decoding Time Deviations
Figure 10: Frame Size Sequences of Sample Streams

(a) Alien

(b) Conan

(c) A Close Shave
Figure 10: Frame Size Sequences of Sample Streams (continued)
Figure 11: Sequences of GOP Decoding Time

(a) Alien

(b) Conan

(c) A Close Shave

32
Figure 11: Sequences of GOP Decoding Time (continued)
5.3 Assumptions

We made the following assumptions for our simulation. First assumption comes from the fact that it is difficult to construct real DVS system. Constructing hardware that supports DVS requires processors whose frequency can be changed 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 [11]. In our experiment, real DVS system is simulated by scaling the frame rate. Every stream has fixed frames per second. If we double the frames per second, 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 as scaling frames per second does not scale the memory access time. This assumption is valid as memory operation time in MPEG decoding is relatively small compared with CPU operation time [37].

Secondly, we exclude displaying in our simulation, because the state-of-the-art technology executes those routines on other devices such as video card. Parsing and minor operations are also ignored as they take less than 3 percent of total decoding time. Therefore, IDCT and reconstruction, which are the main part of decoding, are considered only in our simulation.

Third assumption is that the machine was considered to use no energy when idle, and to use energy/instruction in proportion to $n^2$ when running at a speed $n$, where $n$ varies between 0 and 1. This is a bit optimistic because a chip will draw a small amount of power while in standby mode, and we might not get a one-to-one reduction in voltage to clock speed. However, the baseline power usages from running at full speed also assume that the CPU is off during idle times.
Fourth assumption is that it takes no time to switch speeds. This is also optimistic. 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 operation frequency. Though this is not true as the voltage approaches threshold voltage, this assumption does not effect much to simulation results. The voltage and frequency are supposed to set linearly not restricted to discrete levels.
VI. Performance Evaluations

We compare four MPEG decoders with the sample streams specified in Table 1. The first one is original MPEG decoder from Boston University without any modification, the second one is MPEG decoder with shutdown mechanism, the third implements DVS-DM, and the last one runs DVS-PD. Though the first, the third and fourth MPEG decoder, are implemented and simulated to obtain energy consumption, the second MPEG decoder with shutdown mechanism is not a real implementation. We calculated energy consumption by shutdown algorithm theoretically with original MPEG decoder assuming that the shutdown occurs immediately when the system is idle, and no energy is consumed on shutdown and wake up. Energy consumption by processor shutdown algorithm is calculated by removing delay time out of the whole decoding time of the original MPEG decoder as in the following equation.

\[ E_{\text{shutdown}} = E_{\text{original}} \times \frac{\text{Decode\ Time} - \text{Delay\ Time}}{\text{Decode\ Time}} \]

where \( E_{\text{shutdown}} \) is the energy consumption by the shutdown algorithm and \( E_{\text{original}} \) is by the original MPEG decoder. \( \text{Decode\ Time} \) is total decode time by original MPEG decoder and \( \text{Delay\ Time} \) is the total time the decoding is delayed in DELAY_PHASE as explained in Chapter 3. This is reasonable because the original decoder busy waits when the system should be delayed to meet realtime constraints while decoder with shutdown mechanism shuts down the processor instead of busy wait.

Our simulation shows that MPEG decoder with DVS-PD consumes the least energy among the four decoders and MPEG decoder with DVS-DM is slightly better than shutdown algorithm except Charlie’s Angel. Figure 12
shows the energy consumption by MPEG decoders with six sample streams when the energy consumption of original MPEG decoder is set to 1.

Looking more closely over our results, the difference between performances of each decoder is related with average GOP decoding time. High average GOP decoding time means there is small room for energy saving: if average decoding time is too high that system cannot catch up, no energy saving may occur.

The played frame ratio to the passed frame represents the QoS of MPEG decoding. In our simulation, the played frame ratios by all strategies are above 95% as displayed in Figure 13. The QoS of MPEG decoding is closely related with average decoding time: the QoS reduction of Alien and A Close Shave, which have long average decoding time is large in both decoder with DVS-DM and decoder with decoding time prediction. Overall QoS in decoder with DVS-
DM is better than decoder with DVS-PD. This is due to the inherent characteristic of DVS-DM algorithm: DVS-DM tries to minimize drop rate.

![Figure 13: Played Frame Ratio](image)

### 6.1 Decoder with Shutdown mechanism

It is observed that the energy consumption by decoder with shutdown algorithm is proportional to the average GOP decoding time. It is not surprising because energy in shutdown algorithm is calculated by the GOP decoding time and delay time.

Energy consumption in wake up time is not considered, but it is not ignorable in real system as it takes not only energy but also time. At the wake up time, other devices should wait consuming energy. Played frame ratio is displayed as equal to the original MPEG decoder assuming that no latency occurs when CPU wakes up. But, in real case, the wake up latency is long compared with latency when scaling voltage. Therefore, QoS in real system can be reduced in real system.
6.2 Decoder with DVS-DM

The energy consumptions of this decoder is slightly better than decoder with shutdown mechanism except Charlie’s Angel. Low saving on Charlie’s Angel is due to the high fluctuation. Overall energy consumption is proportional to the average decoding time and effected by fluctuation. High fluctuation makes it difficult to predict the workload of next GOP because it depends on previous workload.

QoS is related with fluctuation and deviation of GOP decoding time as high fluctuation often leads this algorithm to under-estimate of next GOP decoding time.

6.3 Decoder with DVS-PD

The high energy saving on decoder with DVS-PD comes from the fact that the workload of next GOP is predicted by both previous workload and decoding time prediction, thus prediction for next GOP workload is more closer to the real workload than other approaches. To clarify this, we introduce error rates of decoding time predictor which is calculated by following equation.

\[
\text{Error Rate} = \sqrt{\frac{\sum (\text{est}_{\text{dec}} - \text{decode time})^2}{\text{avg}_{\text{GOP}} \text{ time}}}
\]

where \(\text{est}_{\text{dec}}\) is estimated GOP decoding time, \(\text{decode time}\) is real decode time and \(\text{avg}_{\text{GOP}} \text{ time}\) is average GOP decoding time.

Error rates of sample streams are displayed in Figure 14. By observing the
error rates, we can conclude that fluctuation is not much related with error rate. This means that workload prediction of next GOP is not much affected by fluctuation. As a result, energy consumption by this algorithm is not much related with fluctuation: average GOP decoding time is the dominant factor of energy consumption.

![Figure 14: Error Rates of Sample Streams](image)

It is obvious that high error rates causes skipping many frames on that GOP decoding. Thus, in addition to the average decoding time, error rate causes the reduction of QoS in this decoder. A Close Shave which has highest error rate has the poorest QoS. The overall played frame ratio is slightly worse than DVS-DM.

Though this scheme performs best in our simulation, there is a room for improvement. Both energy saving and QoS is observed to be related with error rate, which mean more accurate predictor of MPEG decoding time can reduce error rate and uplift the performance.
VII. Conclusions

Dynamic voltage scaling is a powerful methodology for reducing power consumption of processor. But it is difficult to apply this method to MPEG decoding due to variability of workload, while it is one of the most important but power consuming application in mobile devices.

In this thesis, we present two DVS algorithms on MPEG decoding. One is DVS-DM (DVS with delay and drop rate minimizing algorithm) which relies on data from previous workload of MPEG decoding. In this algorithm, voltage is scaled down in proportion to the delay value which represents the system is too fast. On the contrary, if drop rate is set to non zero, which means system is too slow to decode all frames, voltage is scaled up according to the drop rate. Another algorithm is DVS-PM (DVS with prediction of MPEG decoding time) which uses both previous workload and predicted MPEG decoding time. MPEG decoding time can be predicted by frame sizes of a GOP.

We compare four MPEG decoders with six sample streams. The first one is original MPEG decoder from Boston University without any modification, the second one is MPEG decoder with shutdown mechanism, the third implements DVS-DM, and the last one runs DVS-PD. From our simulation, we found that MPEG decoder with DVS-PD shows the best performance with respect to energy consumption and MPEG decoder with DVS-DM is slightly better than shutdown algorithm except Charlie’s Angel. The high energy saving on decoder with DVS-PD comes from the fact that the workload of next GOP is predicted by both previous workload and decoding time prediction, thus prediction for next GOP workload is more closer to the real workload than other approaches.

Energy saving is closely related with average decoding time. High
workload makes less room for energy saving. In decoder with DVS-DM, high fluctuation makes it difficult to predict future workload by previous workload and causes low efficiency. On the contrary, decoder with DVS-PD is not much affected by the fluctuation. Instead, the performance of DVS-PD depends on the error rate of the predictor.

The QoS of MPEG decoding in both DVS-DM and DVS-PD has relationship with average decoding time. Long average decoding time provides more chance to drop frames. In DVS-DM, the QoS is affected by fluctuation while error rate causes reduction of QoS in DVS-PD. We found that both energy saving and QoS in decoder with DVS-PD, is related with error rate. Thus, implementing more accurate predictor of MPEG decoding time can enhance the performance of the decoder.
석사학위논문

MPEG 디코딩에서의 동적 전압 조정

공학부 손동환

에너지 효율은 이동컴퓨팅의 성장과 더불어 현재 컴퓨팅에서 가장 중요한 요소 중 하나이다. 특히 이동기기의 프로세서에서의 에너지 소모는 이동컴퓨팅 작업이 복잡해지고 내장형 시스템의 증가로 인하여 전체 시스템의 에너지 소모의 많은 부분을 차지하고 있다. 프로세서의 에너지 소모를 줄이는 가장 효율적인 방법은 전압을 낮추는 것인데 이러한 전압의 감소는 프로세서 속도의 감소를 가져오므로 일정 전압으로 낮추는 대신 시스템 작업량에 따라 동적으로 전압을 조정하는 방법에 대한 많은 연구가 이루어지고 있다.

한편 이동컴퓨팅의 어플리케이션에서 점차 멀티미디어의 비중이 커지고 있으며 그 핵심에 있는 MPEG 디코더는 많은 프로세서 파워를 요구하고 따라서 전력소모가 큰 어플리케이션이다. MPEG 디코더의 특징은 작업량의 변화가 크다는 것인데 이로 인하여 이전 단계의 작업량에 기초하여 다음 단계에 대한 작업량을 예측하는 것이 어렵기 때문에 이에 기반한 동적 전압 조정은 효율적이지 않다. 하지만 MPEG 프레임 크기를 통한 디코딩 시간 예측이 가능하다.

이 논문은 MPEG 디코딩에서의 동적 전압 조정 알고리즘을 기존 MPEG 디코더에 구현 및 시뮬레이션 하여 에너지 소모를 비교하는
것을 목적으로 한다.

두 가지 알고리즘이 제안되었는데, 첫번째 알고리즘은 디코딩을 수행하는 동안 이전 단계의 작업량을 나타내는 delay 와 drop rate 를 이용하여 다음 단계에서의 전압을 결정하는 알고리즘이고, 두 번째 알고리즘은 MPEG 프레임 사이즈를 이용하여 디코딩 시간 예측을 통해 전압을 조정하는 것이다. 이 알고리즘들은 Boston University 에서 구현한 MPEG Decoder 에 적용이 되었고 여섯 개의 영화 스트림을 통해 실험되었다. 나타난 결과에 의하면 디코딩 시간 예측을 통한 전압 조정이 다른 알고리즘에 비해 많은 에너지 절약을 가져왔고 에너지 소모가 디코딩 시간 예측의 정확도에 영향을 받는다는 사실을 확인할 수 있었다.
References


[27] LongRun technology from Transmeta
http://www.transmeta.com/crusoe/lowpower/longrun.html


[35] MPEG Decoder from Boston University