Workload Prediction and Dynamic Voltage Scaling for MPEG Decoding

Ying Tan Parth Malani Qinru Qiu Qing Wu
Department of Electrical and Computer Engineering
State University of New York at Binghamton
Binghamton, NY 13902-6000
E-mail: {ying, parth, qqiu, qwu}@binghamton.edu

Abstract – In this paper we present three efficient DVS techniques for an MPEG decoder. Their energy reduction is comparable to that of the optimal solution. A workload prediction model is also developed based on the block level statistics of each MPEG frame. Compared with previous works, the new model exhibits a remarkable improvement in accuracy of the prediction. The experimental results show that, with the new prediction model, the presented DVS techniques achieve more energy reduction than previous works while delivering the same Quality of Service (QoS).

I. Introduction

The ever increasing computing power of battery operated portable devices opens a new era for mobile multimedia applications. It is important to develop techniques to reduce the energy dissipation of such applications so that the life time of the battery can be extended [1]-[5]. One of the representative examples of multimedia application is MPEG decoding. The processing time of MPEG decoding varies significantly due to different frame types and variation between scenes. We call this processing time as workload. *Dynamic Voltage Scaling (DVS)*, which allows the processor dynamically alter its speed and voltage at run time, is one of the most popular energy reduction techniques for the applications that have large workload variations [9].

Using DVS will impact the QoS of the MPEG decoder in several ways. The first to consider is the frame dropping rate. The decoder displays decoded frames at a constant rate. Each frame must be decoded before its display deadline. Otherwise, it will be dropped. The DVS algorithm has the potential to intensify frame dropping. Another impacting factor is the buffer size. Buffers are usually used with DVS to even the workload. Input buffers and output buffers can be inserted before and after the MPEG decoder. They provide the opportunity to "borrow" or "steal" processing time among adjacent frames so that a constant voltage can be used for decoding [1][4][5]. However, increasing the buffer adds the hardware cost. Careful trade-off decision should be made. Finally, the decoding time for each frame is different when using DVS, however, the frame input and display rate remain constant. To guarantee a smooth and continuous display, the decoding and displaying of the first frame need to be delayed, which we refer as *decoding latency*. Input buffers are needed to store the incoming frames before they entering the decoder. The buffer size is proportional to the length of the latency.

In this paper, we measure the quality of DVS strategy with its energy reduction, the buffer usage, the frame dropping rate and the decoding latency. Three DVS schemes for MPEG decoding are proposed, all of which achieve comparable energy reduction as the optimal solution.

Global-Grouping is an offline algorithm whose energy consumption is on average the most close to that of the optimal solution, i.e. decoding all the frames on the lowest possible and constant speed, among all three proposed approaches. With certain decoding latency and some input/output buffers, the Global-Grouping guarantees a continuous display at a constant rate without frame dropping, provided that the workload information of each frame is accurate. Two online heuristic algorithms, *Dynamic-Grouping* and *GOP-optimal*, are also proposed with different energy reductions and buffer requirements.

For most DVS techniques to achieve a good performance, it is important to predict the workload of each task as accurate as possible. In this paper we develop a linear model to predict the decoding workload of each frame. To the best of our knowledge, this is the first prediction model that penetrates into the layered structure of video stream and utilizes the information lying at block level instead of frame level or macro block level. It gives more than 50% reduction in prediction error compared with some of the best known approaches.

The rest of this paper is organized as follows. Section II describes the background of MPEG and related works in this area. In Section III we discuss our prediction model and its implementation in detail. Our scheduling methods are given in Section IV. We represent our experimental results and discussion in Section V. Finally, the conclusions are given in Section VI.

II. Background and related works

A. Background of MPEG

MPEG is a video compression standard which represents the video stream as a series of still images [3]. These images, also called frames, are displayed sequentially at constant rate (e.g. 25 fps or frames per second). There are three types of frames defined in MPEG standard. **I**-frames or *intra-coded* frames are encoded as a whole image i.e. it does not depend on any other picture. **P**-frames or *predictive coded* frames are encoded using past I or P frame as a reference. Finally there are **B**-frames also called as *bi-directionally predictive coded* frames which use both past and future I or P frames as references. The MPEG encoder always sends the encoded frames in a rearranged order so that the MPEG decoder can decompress the frames with minimum frame buffering [7]. For example, a movie with frame order of IBBPBBP will be rearranged in the output sequence as IPBBPBB.

The MPEG video stream has a hierarchical layered structure. From top to bottom, it can be divided into sequence, GOP, frame, slice, macro block and block layers. A video stream is a *sequence* of GOPs (*Group of Pictures*), each one of which comprises of several frames (ideally 12 to 15). Each frame is further divided into vertical strips called slices. Each slice contains several macro blocks which are a 16 by 16 pixel area of the image. There are six blocks per macro block amongst which four are luminance (Y) and two are chrominance (Cr and Cb) blocks.

There are different types of macro blocks similar as frames. *I macro blocks* are encoded without using any other macro block as a reference. *P*, *B* and *Bi macro* blocks are encoded with forward, backward and bi-directional references respectively. Further, there can be different types of macro blocks within a single frame. The I frame contains I macro blocks only. The P frame contains both I and P macro blocks and the B frame contains all of the four types of macro blocks.

Three major operations in MPEG decoding, which consumes most of the processing time, are Run Length Decoding, Inverse Discrete Cosine Transform (IDCT) and motion compensation. All of the four types of macro blocks require Run Length Decoding during their decoding. I macro blocks also require IDCT. P, B and Bi macro blocks may require IDCT and in addition also require motion compensation.

A detailed study of the MPEG coding algorithm shows that the matching process in motion estimation, which is the counter part of motion compensation at the encoder side, is done at *block* level. For example, in process of decoding a P macro block there can be a block which does not require IDCT and decoded only using motion compensation while the other blocks require both. This is the major motivating factor for our prediction model.

B. Related Works

Generally, previous works on DVS for MPEG decoding can be classified into two categories: prediction-based and non-prediction-based.

For the prediction-based scheduling, the accuracy of predicted workload plays a significant role in the performance of these techniques, either for energy saving or for QoS. Most prediction mechanisms utilize the correlation between the frame size and the frame decode time [2][8][9]. A linear relation is usually depicted between these two. The authors of [9] developed three predictors to predict decoding workload of each frame. The best one, which will be denoted as Frame_Type_Len in the rest of the paper, dynamically updates the average decode time for each frame type and then adds an offset using a weighted factor based on the slope of frame size vs. decode time curve. Our experimental results show that, by carefully analyzing the input video stream, our predictor gives more accurate results than the Fram_Type_Len predictor.

The prediction accuracy can be improved by considering other variables lying in a video stream apart from frame size and types. The authors of [1] divide the frame decoding time into two parts, frame-dependent (FD) part and frame-independent (FI) part. The time of the FD is predicted as the moving average of previous FD. The decoder in this work is assumed to decode only one frame in each display interval, which limits its energy saving.

There are also some improved DVS techniques that do not rely on workload prediction [4][5]. In both of the two works, buffers are used to avoid deadline missing. Reference [5] introduces an online DVS technique that fully utilizes the VST (workload-variation slack time) of each task. However, the worst case execution time is assumed to be known in advance, which is not very practical in real world. Another DVS technique using feedback control [4] to adjust the supply voltage based on the number of the frames in the output buffer. However, they assume that a frame is always ready for decoding. Furthermore, it is difficult to control the gain of the feedback controller and a slight change in the gain has a great impact on the entire performance. Despite of the above mentioned limitations, it is still the best existing technique we are aware of for both energy reduction and deadline missing control. In Section V, we will compare our algorithms with this approach.

III. Workload prediction

The decode time of each frame varies primarily with the frame size. However, considering only frame size to estimate the workload leads to poor prediction accuracy. Another operation in decoding process which is responsible for introducing workload variation is IDCT. We experimentally developed a prediction model based on the number of IDCT computations required for each frame. This prediction scheme yields better results but still suffers from poor correlation.

We carefully analyzed the MPEG encoding/decoding algorithm and realized that different blocks inside the same macro block may require different decoding operations (i.e. IDCT and motion compensation). Some of them may need only motion compensation while others need both. It is because the matching process in motion estimation at the encoder side is done at block level and some blocks have a zero remaining energy after motion estimation (not requiring a DCT operation) while others have a non-zero remaining energy.

TABLE 1 Relation between the macro blocks and the blocks

Block Type \ Macro Block		I	P	B	Bi
IDCT only	**M_1**	X	X	X	X
IDCT + FW Motion	**M_2**		X		
FW Motion only	**M_3**		X		
IDCT + BW Motion	**M_4**			X	
BW Motion only	**M_5**			X	
IDCT + Bi Motion	**M_6**				X
Bi Motion only	**M_7**				X
No IDCT No Motion	**M_8**		X	X	X

The processing times for forward, backward and bi-directional motion compensation are different. For example, the bi-directional motion compensation takes the longest time because it needs to consider two references. Finally, in P, B and Bi macro blocks, there are a large number of skipped blocks which are copied directly from the reference block. No IDCT or motion compensation is needed for these blocks. The processing time for these skipped blocks is simply the time for memory read and write. Based on the above observations we divide the MPEG blocks into 8 different types given in the first column of TABLE 1. Different types of blocks require different processing during the decoding. Not all of the 8 types of blocks can co-exist in a macro block. The relation between the macro blocks and different types of blocks are summarized in TABLE 1. The variable M_i, $1 \le i \le 8$, is used to represent the number of type i blocks in a frame. The method to extract these values will be discussed later.

Our analysis shows that a considerable variation still exists for the decoding time of the I frames, although they have the same number of type 1 blocks. It means that the processing of the Run Length Decoder is not negligible. Further study shows that the processing time of the Run Length Decoder is proportional to the size of the data. Therefore, another variable, M_9, is introduced to account for the size of the frame.

Equation (1) shows our prediction model based on these nine variables. The coefficients w_0, w_1, …, w_9 are obtained using linear regression analysis.

$$frame\_decode\_time = w_0 + \sum_{1 \le i \le 9} w_i \cdot M_i \qquad (1)$$

For the same MPEG decoder, the processing time of Run Length Decoding, IDCT and motion compensation on a single block is the same for different types of movies. With the above formulation of the frame decoding time, we can derive one set of regression coefficients that works for all types of movies. Hence, only one predictor is needed for each decoder. Most regression analysis based prediction model need to have several sets of regression coefficients for different types of movies.

We applied our prediction model to a variety of movies including animated movies, high motion and low motion scenes from actual movies. The length of these movie clips ranges from 150 frames up to 3000 frames. The target decoder is the Berkeley MPEG decoder [7] running on Pentium IV 2.6GHz processor. We simulated and compared our prediction results with that of two other predictors given by reference [1], which to the best of our knowledge are the most efficient amongst all contemporary approaches. The Frame_Avg approach predicts the decoding time as the moving average of previous decoding time for each type. The Frame_Type_Len approach improves the Frame_Avg approach by adding an offset to account for the frame size. The Frame_Type_Len approach is also trained with the same set of movies to obtain a fair comparison.

Fig. 1 depicts the comparison of the Prediction Errors (PE) of three predictors.

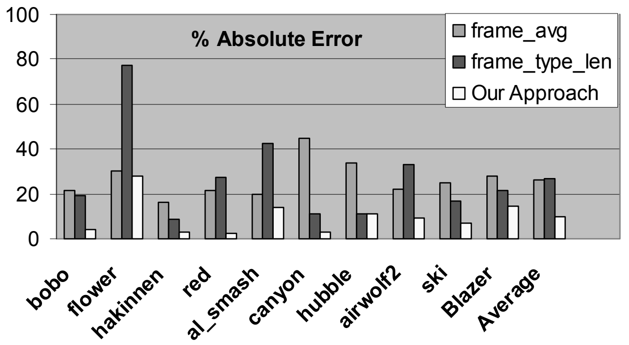
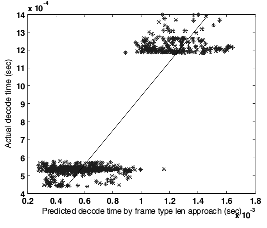


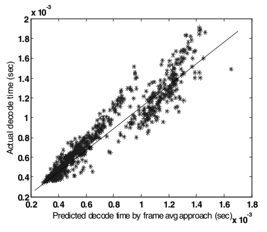
Fig. 1. Comparison of prediction errors

The PE is calculated as $PE = \frac{1}{n}\left|\frac{T_p - T_a}{T_a}\right|$, where T_p and T_a are predicted and actual decoding time respectively. As can be seen from the figure our predictor has an average of 66% improvement in the prediction error compared to both approaches.

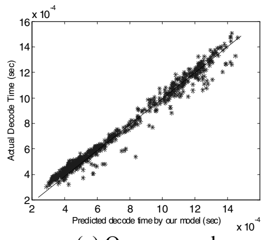
In addition to the lower prediction error, the new predictor gives better correlation with the actual value. Fig. 2 gives the scatter plot of the predicted workload vs. actual workload for those three approaches. Fig. 3 gives the comparison of the correlation coefficients.



(a) Frame_Avg



(b) Frame_Type_Len



(c) Our approach

Fig. 2. Predicted workload vs. actual workload

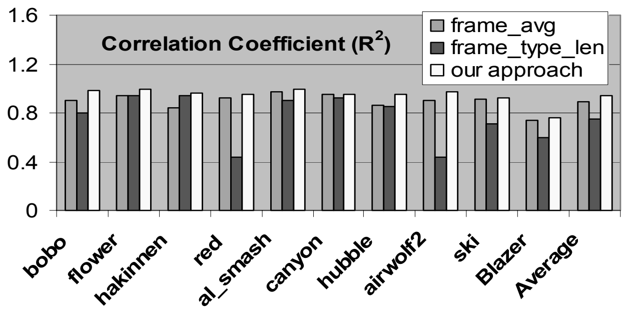


Fig. 3. Comparison of correlation coefficients

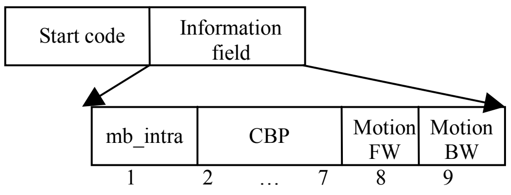


Fig. 4. Format of macro block header

At the last of this section, we go back to the layered structure of MPEG video stream and explain how to extract the variables M_1~M_8. There is a fix sized header at each hierarchy level of a MPEG stream, which contains important information needed for decoding. The information used by our model resides in the macro block header which contains a *start code* and 6 bits of information. The value of these bits is used as an index for a look up table stored in the decoder and each 9 bits entry in this table contains four fields as illustrated in Fig.4. The flag *mb_intra* indicates whether the given macro block is intra coded or not. The *CBP* (coded block pattern) is a six bit code that indicates whether the IDCT operation is needed for each block. The *Motion FW* and *Motion BW* flags indicate the existence of forward and backward motion compensations. By combining these parameters we can easily count the number of each categorized block for our model.

IV. DYNAMIC VOLTAGE SCALING

In this section, we introduce three DVS methods for MPEG decoding. For these approaches, continuous frequency/voltage scaling capabilities are assumed. Also, since the time needed to switch between different voltage settings is less than 1% of the time needed for decoding each frame [1], and some newer processors may even lower this percentage, we assume that the switch time is negligible. It is also assumed that the input and display of the MPEG decoder are at a constant rate whose period is *T*.

Given a MPEG stream with *n* frames, the optimal constant decoding voltage V_{opt} can be calculated as:

$$V_{opt} = \frac{\sum_{i=1}^{n} load_i}{i} \cdot \frac{V_{full}}{load_{\max}},$$

where $load_i$ is the workload of the *i*th frame, $load_{max}$ is the largest workload and V_{full} is the level of supply voltage which finishes processing the $load_{max}$ within T . Using the V_{opt} the video stream can be decoded within $n*T$ time and the energy dissipation is minimal. We call this approach as optimal-VS.

There are several limitations with the optimal-VS. 1) It requires that the workload of each frame is known in advance 2) It decodes the frame at a constant speed without considering the frame incoming time or output deadline. As a result a large input/output buffer is needed; otherwise, there will be a QoS penalty. The following three DVS algorithms are developed to resolve these two problems.

A. GOP-optimal DVS

The GOP-optimal algorithm buffers all the frames in a GOP, estimates their workload, and decodes the entire GOP using a constant voltage that is calculated similar as V_{opt}. This is an on-line heuristic that is based on Optimal-VS. It does not need the workload information of the entire video stream before decoding. However, it does not consider the frame incoming time and display deadline either. The first frame in a GOP is always an I frame, which usually has larger workload than other frames in the GOP and needs longer decoding time.

To guarantee that there is always a frame ready to be displayed, the simplest way is to start displaying the first frame after the entire GOP has been decoded. As a result, the output buffer of the decoder should be large enough to hold all of the frames in a GOP. Furthermore, the input frames come in at a constant rate. Because the first frame in a GOP takes the longest decoding time, extra buffers are needed to store the incoming frames. In the worst case, the input buffer needs to be two GOP long. Note that these are only conservative estimations of the buffer size. The actual buffer usage could be less.

B. Global-Grouping

Let the display time of the *i*th frame denoted as D_i. The time to decode a *n* frame MPEG stream can be divided into *n* intervals $(0, D_1)$, (D_1, D_2), …, (D_{n-1}, D_n). The Global-Grouping gathers consecutive intervals into groups. It divides the decoding time into *m* groups G_1, G_2, …, G_m. Inside the *j*th group, a constant voltage V_j is used. V_j is selected such that all frames whose display time is within G_j can be decoded before its display deadline. Note that the Global-Grouping is applicable to the general applications with deadlines.

During the grouping procedure, the average workload in the time zones (D_0, D_1), (D_0, D_2), … , (D_0, D_n) will be tested and the intervals in the time zone that has the maximum average workload will be grouped together. After that a new search will start on the rest of the intervals. The pseudo code of the Global-Grouping algorithm is given in Fig. 5, where C_i is the decoding workload of frame *i*, which should be displayed at time D_i.

As mentioned in section II, a B frame has two reference frames. Both of them are received before the B frame and they also should be decoded before the B frame.

```
1.  Set size as the total number of frames;
2.  index = 0, j = 0;
3.  while index < size
4.      for k = index+1 to k = size
5.          find out the value of k, which makes
            $\frac{\sum_{i=index}^{k} C_i}{D_k - D_i}$ maximum;
6.          make workload from index+1 to k group $G_j$;
7.      end
8.      index = k, j++, return to step 3;
```

Fig. 5. Global-Grouping algorithm

TABLE 2 WORKLOAD REARRANGEMENT

index	1	2	3	4	5	6	7	8
receive order	I_1	P_2	B_3	B_4	P_5	B_6	B_7	I_8
display order	I_1	B_3	B_4	P_2	B_6	B_7	P_5	I_8
Workload (C_i)	I_1	P_2+B_3	B_4	0	P_5+B_6	B_7	0	I_8

While the forward reference frame is displayed before the B frame, the backward reference frame is display after the B frame. Therefore, care should be used when calculating the workload C_i. TABLE 2 shows the relationship of receiving order, display order and rearranged decoding workload in each display interval. Note that the Global-Grouping enables the decoder to borrow the time from the previous interval; therefore the time to decode C_i could be larger than one T.

The result of Global-Grouping has some interesting characteristics as Theorem 1 and 2 states. The proof is straightforward and will be skipped due to the space limitation.

Theorem 1: For each group G_j, we can always find a constant voltage V_j, under which the decoding of each frame can be finished before its display deadline and the idle time of the processor is 0. V_j is monotonically decreasing as j increases and it is proportional to the average workload in this group: $C_{avg} = \frac{\sum_i C_i}{\sum_i (D_i - D_{i-1})}$, where $(D_{i-1}, D_i) \in G_j$.

Theorem 2: The last frame of a group has the largest workload amongst the frames within this group. Its decoding is finished right at its display deadline. Other frames are decoded before their display deadline provided that the workload information is accurate.

Input and output buffers are needed for the Global-Grouping to guarantee that there is always a frame available for decoding or displaying. An example is illustrated in

Fig. 6. In order for each frame to be ready before their decoding starts, the receiving procedure must start $3T$ earlier than the decoding procedure. A buffer at the input side must be used to store the incoming frames during this $3T$ delay period. Therefore, the size of the buffer is 3 frames. Furthermore, since there is a *displaying lag*, output buffers are needed to store those that finish decoding before their display deadline. In the example, a buffer of 2 frames is needed. It is obvious that the input buffer size (*IB*) and the output buffer size (*OB*) always have the following relation $IB = OB \pm 1$.

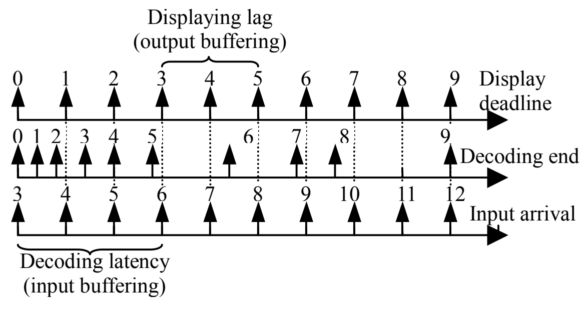


Fig. 6. Input and output buffering

The IB can be calculated as the following equation

$$IB = \max_j[\max_i(i - floor(\sum_{k=start\_of\_G_j}^{i} \frac{C_k}{C_{avg,j}}))] , \quad (2)$$

where $C_{avg,j}$ is the average workload of group G_j. The experiments show that, although the Global-Grouping needs fewer buffers than the Optimal-VS, the amount is still quite considerable.

The Global-Grouping is an offline strategy and it needs the workload profile of the entire video stream. Compared with the optimal solution, it requires less buffer while achieves a comparable energy reduction with less display latency. This algorithm is useful if the processor decodes and displays certain movie clips repeatedly. It can also be applied at the encoder side where the workload profile is available given the condition that the encoder can communicate with the decoder about the grouping and voltage selection results.

C. *Dynamic-Grouping*

The Dynamic-Grouping is an online heuristic based on Global-Grouping. It buffers the input frames up to a certain window size (e.g. a GOP size). The workload of each frame inside the window is predicted and grouping is applied within this window. When a new frame comes in, the decoder first predicts its workload $load_i$ then updates the grouping dynamically as described in

Fig. 7. Here, M is the number of groups in current window and $C_{avg,j}$ is the average workload of the jth group.

The size of the input buffer for the Dynamic-Grouping is equal to the size of the window while the size of the output buffer can be calculated using equation (2). Compared with GOP-optimal, Dynamic-Grouping gives better trade-off between energy and buffer size.

```
1.  $M$ = index of the last group in the current window;
2.  set the incoming frame to be group $M+1$;
3.  for $l = M+1$ to $1$
4.      if ($C_{avg,l} > C_{avg,l-1}$)
5.          merge group $l$ and $l-1$
6.          set the index of the new group as $l - 1$
7.      else
8.          stop
9.  end
```

Fig. 7. Dynamic group update

V. EXPERIMENTAL RESULTS AND DISCUSSION

The presented DVS algorithms are tested on several different movie clips. The statistics of these clips is given in TABLE 3.

We simulated and compared the proposed DVS algorithms with three other algorithms. *1*) Frame-based, which decodes one and only one frame in each display interval, *2*) Feedback control based [4] and *3*) Optimal-VS.

TABLE 3 CHARACTERISTICS OF MPEG CLIPS

MPEG clips		Frame type	# of frames	GOP size
name	index			
hakkinen	1	I,P,B	799	12
bobo	2	I,P,B	679	90
ski	3	I,P,B	1513	15
blazer	4	I,P,B	2998	12
wg	5	I,P	130	6

The energy values are reported as the percentage degradation over the optimal-VS approach. The buffer sizes are the size of display buffer in the unit of frames. Note that for the Global-Grouping and Optimal-VS, the input buffer size is the same as that of the output buffer plus or minus 1. For Dynamic-Grouping and GOP-optimal, input buffers of at most one GOP and two GOPs long are used, respectively. Some input buffers are also needed in feedback approach to guarantee that there is always a frame available to decode whenever the previous one finishes decoding. For all algorithms, the decoding latency is proportional to the input buffer size.

TABLE 4 gives the energy and buffer usage of different DVS algorithms when the workload prediction is perfect. It shows that while the Global-Grouping always gives similar energy reduction as the optimal one, it does not have much reduction in the buffer requirements. The Dynamic-Grouping gives the best balance between the energy reduction and the buffer requirements.

TABLE 5 gives the energy and the buffer usage when the workload prediction is not perfect. The workload prediction is given by our prediction model discussed in Section III. The accuracy of prediction has a great impact on the performance of DVS techniques, in terms of deadline missing and buffer usage, however, not so large impact on the energy dissipation. Since a frame will be dropped if it misses the decoding deadline, this causes unfair energy comparison. We scaled up the predicted workload by 5%, which is enough to make sure that the deadline miss rate is zero. For some cases, this scale increases the buffer usage however decrease the energy dissipation. The results show that both Global-Grouping and Dynamic-Grouping are pretty robust when working with our workload predictor.

TABLE 4 ENERGY AND BUFFER USAGE OF DVS ALGORITHMS WITH PERFECT WORKLOAD PREDICTION

MPEG clips		1	2	3	4	5
Frame-based	Energy (%)	200.3	65.1	88.6	118.2	55.8
	Buffer	1	1	1	1	1
Feedback	Energy (%)	3.3	18.1	20.2	13.4	17.7
	Buffer	9	10	9	10	9
GOP optimal	Energy (%)	1.4	4.5	11.8	5.6	10
	Buffer	3	9	3	5	2
Dynamic grouping	Energy (%)	2.2	2.4	10.1	4.1	10
	Buffer	3	12	4	2	2
Global grouping	Energy (%)	2.1	2.0	2.1	0.5	9.8
	Buffer	8	15	71	56	7
Optimal	Energy (%)	0	0	0	0	0
	Buffer	26	26	97	77	9

TABLE 5 ENERGY AND BUFFER USAGE OF DVS ALGORITHMS WITH IMPERFECT WORKLOAD PREDICTION

Clips	Global grouping		Dynamic grouping	
	Energy (%)	Buffer size	Energy (%)	Buffer size
1	0.78	32	2.8	3
2	0.99	18	1.4	12
3	2.7	79	11.8	5
4	0.3	66	10.7	7
5	0.4	9	4.6	4

VI. CONCLUSIONS

In conclusion of our work, we present a workload prediction model, which is motivated by detailed analysis of MPEG decoding procedure. The predictor utilizes the block level statistics of each MPEG frame and gives highly accurate prediction results. Three DVS algorithms are further presented. All of which gives comparable energy reduction as the optimal voltage scaling and work robustly with our predictor. The experimental results show that the Dynamic-Grouping algorithm gives the best trade-off between energy reduction and the quality of decoding.

REFERENCES

[1] K. Choi, K. Dantu, W. Cheng and M. Pedram, "Frame-based dynamic voltage scaling for a MPEG decoder," *ICCAD'02 – A give the CM/IEEE Int'l Conf. on Computer Aided Design, 2002, pp. 732-737*

[2] M. Mesarina and Y. Turner, "Reduced energy decoding of MPEG streams," *Proc. of Multimedia Computing and Networking, San Jose, CA 2002.*

[3] D. Son, C. Yu, and H. Kim, "Dynamic voltage scaling on MPEG decoding," *International Conference of Parallel and Distributed System (ICPADS),* June 2001.

[4] Z. Lu, J. Lach, M. Stan, and K. Skadron, "Reducing multimedia decode power using feedback control," In *Proceedings of the 21st International Conference on Computer Design (ICCD'03)*, 2003.

[5] C. Im, S. Ha, and H. Kim, "Dynamic voltage scheduling with buffers in low-power multimedia applications," *ACM Transactions on Embedded Computing Systems,* Vol. 3, pp 686-705, November 2004.

[6] Y. Lu, L. Benini, and G. D. Micheli, "Dynamic frequency scaling with buffer insertion for mixed workloads," *IEEE Transactions on computer-aided design of integrated circuits and systems,* 21(11), pp. 1284-1305, November 2002.

[7] http://bmrc.berkeley.edu/frame/research/mpeg/mpeg_overview.html

[8] E. Nurvitadhi, B. Lee, C. Yu and M. Kim, "A comparative study of dynamic voltage scaling for low-power video decoding," *Int'l Conf. on Embedded Systems and Applications,* June 23-26, 2003.

[9] A. Bavier, A. Montz, and L. Peterson, "Predicting MPEG execution times," SIGMETRICS / PERFORMANCE '98, *Int'l Conf. On Measurement and Modeling of Computer Systems, 1998, pp. 131-140.*

[10] L. Benini, G. De Micheli, "System-level power optimization: techniques and tools," *International Symposium on Low Power Electronics and Design*, 1999.