A Workload Prediction Model for Decoding MPEG Video and its Application to Workload-scalable Transcoding

Yicheng Huang, Vu An Tran, and Ye Wang
School of Computing
National University of Singapore
3 Science Drive 2, Singapore 117543
{huangyic,tranvuan,wangye}@comp.nus.edu.sg

ABSTRACT

Multimedia playback is restricted by the processing power of mobile devices, and in particular, the playback quality can be degraded due to insufficient processing power. To address this problem, we propose a new workload-scalable transcoding scheme which converts a pre-recorded video bitstream into a new video bitstream that satisfies the device's workload constraint, while keeping the transcoding distortion minimal. The key of this proposed transcoding scheme lies on a new workload prediction model, which is fast, accurate and is generic enough to apply to different video formats, decoder implementations and target platforms. The main contributions of this paper include 1) a workload prediction model for decoding MPEG video based on an offline bitstream analysis method; 2) a transcoding scheme that uses the proposed model to control the decoding workload on the target device. To facilitate our transcoding scheme, we have proposed a compressed domain distortion measure (CDDM) that takes effects from both frames per second (fps) and bits per frame (bpf) into consideration. CDDM ensures the transcoded video bitstream to have the best playback quality given the device's workload constraint. Both the workload prediction model and the transcoding scheme are evaluated experimentally.

Categories and Subject Descriptors

I.6.4 [**Simulation and Modeling**]: Model Validation and Analysis;
H.5.1 [**Multimedia Information Systems**]: Video

General Terms

Algorithms, Experimentation

Keywords

CDDM, Mobile, Video Transcoding, Workload Prediction Model

1. INTRODUCTION

After a decade of tremendous growth, mobile devices today are becoming an important entertainment platform for streaming video and multimedia content. This application scenario is a fast emerging area with huge economic impact. However, supporting video applications on mobile devices is challenging due to the limited processing power. Advanced video formats such as MPEG-2 and MPEG-4 can effectively compress the video files in terms of bit rate. However, they demand high decoding workload. Currently, most mobile devices' processor frequencies are in the range of 200-600MHz, which makes it hard for them to decode high quality video bitstreams at high frame rates, 20-30 frames per second (fps). For low speed processors, the decoding workload requirement cannot be met. Given versatile mobile devices with various processing powers, how to match a transcoded video bitstream to a mobile device's processing power, so that the best playback quality is guaranteed, is the problem we seek to address in this paper. Clearly, the purpose of our transcoding scheme differs fundamentally from that of conventional transcoding schemes which are usually designed for format conversion or bandwidth adaptation.

Our projected application scenario is that mobile devices request video bitstreams from a server. Due to the limited processing power, mobile devices are not capable of decoding the original video bitstream in real-time. For such a case, we propose a scheme to transcode the original video bitstream to meet the decoding workload constraint of the target device. Figure 1 shows the architecture of our proposed scheme, where a transcoding proxy employing our scheme is setup between the video file server and mobile devices. The proxy receives the architecture-specific information from the mobile devices along with their streaming or downloading requests. According to the provided information, the proxy transcodes the original video bitstream to satisfy the constraint.

The transcoding scheme works in the compressed domain to minimize the transcoding overhead. It does not involve full video decoding and re-encoding, but only drops frames and Huffman codes in the compressed domain. There are two challenges: 1) To decide how many frames or Huffman codes should be dropped so that the reduced workload is kept just below the constraint. 2) To devise an algorithm that selects the best quality video bitstream among all possible candidates with the same workload.

To address the two problems we develop a workload prediction model for video decoding and a compressed domain distortion measure (CDDM). Using the workload prediction

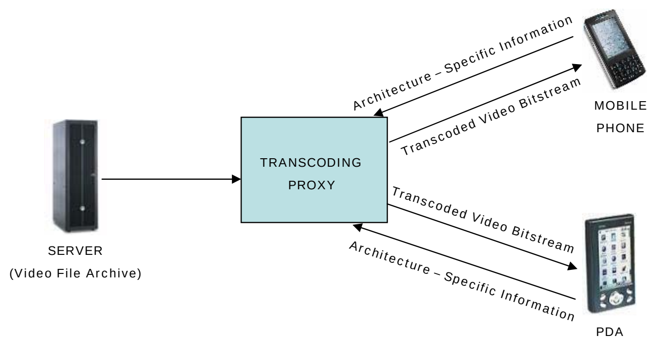
Architecture – Specific Information
MOBILE PHONE
Transcoded Video Bitstream
TRANSCODING PROXY
SERVER
(Video File Archive)
Transcoded Video Bitstream
Architecture – Specific Information
PDA

Figure 1: System architecture for the transcoding scheme

model, we can predict the decoding workload based on information extracted from the video bitstream. The proposed transcoding scheme works in three steps. In the first step, using the workload prediction model, the scheme generates possible candidates satisfying the workload constraint. In the second step, using CDDM, the scheme estimates the distortion of these candidates as compared to the original video bitstream and selects the one with the least distortion. Then, metadata is generated to indicate which frames or Huffman codes should be dropped. In the third step, the transcoder drops frames and Huffman codes according to the metadata and generates the target video bitstream.

The contributions of this paper are two-fold:
1) We propose a workload prediction model that predicts the decoding workload of a video bitstream based on information extracted from the video bitstream. Our experiment shows that the model is very accurate, even with different video formats, decoder implementations and target devices.
2) Based on the workload prediction model, we propose a compressed domain transcoding scheme, which can

- accurately control the decoding workload of the target video bitstream by using the workload prediction model.
- keep the distortion between the target and original video bitstream minimal by using CDDM.

The paper is organized as follows. Related works are reviewed in Section 2. Section 3 details the proposed workload prediction model. The transcoding scheme is presented in Section 4. Evaluation of the workload prediction model and transcoding scheme is in Section 5. The conclusion and future works are discussed in Section 6.

2. RELATED WORKS

Transcoding is a well-studied subject [3, 13, 17, 23]. Earlier transcoders are mainly designed to perform format or bit-rate conversion. In recent years, as mobile devices are becoming a popular multimedia platform, some transcoders are proposed to reduce the frame size or frame rate to fit the constraints of mobile devices. However, to our knowledge, there has been no published work on workload-scalable video transcoding that can accurately match the bitstream to the target mobile device's processing power. We believe that this will become an important application scenario in the near future. This section, therefore, surveys related works on MPEG video workload modeling.

Several workload prediction models for video decoding have been proposed in the literature. The existing models can be classified into two categories: models based on history (online approach at the client side to predict workload on-the-fly based on workload history) and models based on information extracted from the video bitstream (offline approach to extract information from the bitstream to obtain the predicted workload in the form of metadata).

In the first category, Choi et al. [9] have proposed a frame-based Dynamic Voltage Scaling (DVS) scheme. The decoding workload of the current frame is predicted by a weighted-average of workloads of the previous same-type frames. Bavier et al. [7] proposed a model which can predict not only the decoding workload of a frame, but also the decoding workload of a network packet. In that paper, three predictors to predict the workload of decoding a frame and another three predictors to predict the workload of decoding a packet were proposed and analyzed in terms of performance. Son et al. [19] proposed a model that predicts the decoding workload in a larger granularity, Group of Pictures (GOP), which contains a number of frames. This prediction model makes use of previous frames' workloads, and incoming frames' types and sizes. The history-based models need to fully decode the video bitstream to obtain the historical record. Compared to video decoding, the computational complexity of prediction is very low. These models are usually adopted at the client side to predict the workload on-the-fly. However, due to the unpredictability of video decoding workload (the maximum workload of decoding a frame or a macroblock (MB) can be larger by more than ten times of the minimum workload), the history-based models suffer in terms of accuracy.

The models in second category (offline bitstream analysis) predict the decoding workload based on information extracted from the video bitstream. In [14], Mattavelli et al. proposed a scheme that divides the decoder into several tasks and predicts each task by a linear function. The model's parameters are obtained by simulation to build the model. The prediction by using this model does not need full video decoding. Prediction results can be inserted into frame header in any format. However, due to the unpredictability of video decoding workload, estimating video decoding workload by mapping to some linear function will not achieve good accuracy. Our analysis also shows that tasks such as motion compensation (MC) cannot be modeled as a linear function.

For the second category, Lan et al. [12] also proposed a model that predicts the workload of decoding one macroblock by four parameters: macroblock type, motion vector magnitude, motion vector count and number of non-zero DCT coefficients. These parameters are multiplied with corresponding weights and added with a safety margin to get the prediction result. Although this model can predict the decoding workload accurately, it is not designed to apply to generic processors, since the model is proposed for a decoder implemented on a processor that is designed specifically for multimedia processing. It is also unclear about the decisions to select the weights for these parameters.

Schaar et al. [18] introduced a concept of virtual decoding complexity, which can be regarded as a special feature of the video bitstream. For different target devices, the virtual decoding complexity is converted to the actual workload using different parameters. By adding a layer of virtual decoding complexity between the video bitstream and actual workload, this approach can be easily extended to a variety of existing and future image and video compression schemes. However, the computation for the virtual decoding complexity needs information derived from the decoded pixel value. In other words, if we want to compute the virtual decoding complexity of the video, we have to fully decode it first, and this is computationally expensive.

The models in [12, 14, 18] were not evaluated for different decoder implementations and video formats. To our knowledge, different decoder implementations and video formats affect the decoding workload considerably. A model suitable for one decoder implementation or video format may not be suitable for others. Therefore, the models in [12, 14, 18] may not be generic for different decoder implementations and video formats.

In this paper, we propose a new workload prediction model. It predicts the decoding workload based on information extracted from the compressed video bitstream. The proposed model has advantages of being:
1) Accurate: Our experiments show that the model can predict the decoding workload of a frame within an error rate of 2% on average.
2) Generic: The model applies to different video formats, decoder implementations and target devices.
3) Fast: The model only needs information from the compressed domain for prediction, i.e., neither IDCT nor MC is needed during the runtime.

3. A WORKLOAD PREDICTION MODEL FOR DECODING MPEG VIDEO

This section presents a new workload prediction model to predict the decoding workload of video bitstreams. The model is designed to be generic to different video formats, decoder implementations and target devices.

Considering a video bitstream encoded in a typical video format such as MPEG-2 or MPEG-4, it is made up of frames which consist of several slices, which in turn consists of Macroblocks (MBs). Hence, decoding a video bitstream can be considered as decoding a sequence of MBs. In our model, the decoding workload is predicted in the MB granularity. Decoding a MB involves variable length decoding (VLD), inverse quantization (IQ), DC-AC prediction, inverse Discrete Cosine Transform (IDCT), and Motion Compensation (MC). For each task, the workload prediction is done separately and the prediction workload of the whole MB is the sum of all tasks' workload.

3.1 VLD, IQ and DC-AC Prediction Tasks

3.1.1 VLD

In modern video codecs, DCT coefficients are encoded using variable length coding (VLC), which involves run length coding, followed by Huffman coding. The workload of Huffman decoding depends on the number of Huffman codes which is equal to the number of non-zero DCT coefficients. Therefore, the workload of VLD in decoding one MB depends on its number of non-zero DCT coefficients. Experimental results show that the relationship between VLD workload and the number of non-zero DCT coefficients is linear, i.e., $W_{vld} = a_{vld} \times n_{coef} + b_{vld}$, where W_{vld} is the workload, n_{coef} is the number of non-zero DCT coefficients in the MB, a_{vld} and b_{vld} are parameters. The values of a_{vld} and b_{vld} vary for different MB types. We model the VLD task by:

$$W_{vld} = \begin{cases} a_{vld\_intra} \times n_{coef} + b_{vld\_intra}, & \textit{Intra MB type} \\ a_{vld\_inter} \times n_{coef} + b_{vld\_inter}, & \textit{Inter MB type} \\ b_{vld\_skip}, & \textit{Skipped MB type} \end{cases} \tag{1}$$

3.1.2 IQ

For video decoders, there are two typical implementations of the IQ task. The first implementation is to multiply the quantization coefficients with every DCT coefficient. The second implementation, which is more optimized, is to multiply the quantization coefficients only with the non-zero DCT coefficient. For the first approach, the workload of the IQ task can be modeled as a constant parameter C_{iq}, because for one MB, the number of DCT coefficients is fixed. For the second approach, the workload of IQ can be modeled as a linear function of the number of non-zero DCT coefficients, i.e., $W_{iq} = a_{iq} \times n_{coef}$, where W_{iq} is the workload of IQ, n_{coef} is the number of non-zero DCT coefficients in the MB and a_{iq} is a parameter. To adapt to different implementations, we model the IQ task as:

$$W_{iq} = a_{iq} \times n_{coef} + b_{iq} \tag{2}$$

For the first approach, a_{iq} is 0 and b_{iq} is equal to C_{iq}. For the second approach, a_{iq} is c_{iq} and b_{iq} is equal to 0.

3.1.3 DC-AC Prediction

The DC-AC Prediction task is to estimate the DC or AC coefficients from the previous decoded DC and AC coefficients. Experimental results show that for the same MB type, the workload of the DC-AC Prediction task can be approximated to a constant value. Hence, we model the DC-AC Prediction task by:

$$W_{acdc} = \begin{cases} b_{acdc\_intra}, & \textit{Intra MB type} \\ b_{acdc\_inter}, & \textit{Inter MB type} \\ b_{acdc\_skip}, & \textit{Skipped MB type} \end{cases} \tag{3}$$

For some video format such as MPEG-2, there is no DC-AC Prediction task. Our model can also adapt to those cases by setting b_{acdc_intra}, b_{acdc_inter} and b_{acdc_skip} to zero.

Since VLD, IQ and DC-AC Prediction tasks can be either modeled as a linear function of the number non-zero DCT coefficients or a constant function, we can combine the three tasks' models together:

$$W_{vld} + W_{iq} + W_{acdc} = \begin{cases} a_{intra} \times n_{coef} + b_{intra}, & \textit{Intra MB type} \\ a_{inter} \times n_{coef} + b_{inter}, & \textit{Inter MB type} \\ b_{skip}, & \textit{Skipped MB type} \end{cases} \tag{4}$$

where a_{intra}, a_{inter}, b_{intra}, b_{inter} and $b_{skipped}$ are parameters depending on the target platform, video format, and decoder implementation.

3.2 IDCT Task

Each MB consists of six blocks: four Y blocks, one U block and one V block with a size of 8×8 pixels each. The input data to the IDCT task is the same for all MBs, which results in the same computational workload being incurred. However, the decoder may optimize the IDCT's implementation by considering the position of the least important non-zero DCT coefficient to avoid redundant computation [16]. Some decoders even implement the IDCT tasks in different ways for different MB types. To make our model generic to different decoder implementations, we separate the IDCT task into six sub tasks, and each task is the IDCT operation on an 8×8 block. Since MB type can be one of the three types: Intra, Inter, or Skipped, and for one block there are 64 positions of DCT coefficients, the sub task can be modeled as a 3×64 table. The items (values) in the table are the workload of IDCT task for the block with the MB type and the position of the least important non-zero DCT coefficient provided. The workload of IDCT task of a MB is then predicted as the sum of the six sub IDCT tasks.

3.3 MC Task

The MC task is also divided into six sub tasks with each sub task as a MC operation for an 8×8 block. Experimental results show that the workload of the MC task depends on the MB type, MC type and motion vectors' precisions. For one MB, there are at most N motion compensation types ($N < 10$), and its type can be one of the 3 MB types: Intra, Inter, or Skipped. And there are 4 possible precisions for both x-dimension and y-dimension motion vector (one-pixel, half-pixel quarter-pixel, and eighth-pixel precision). Hence, the model for a sub MC task is a table of size $3 \times N \times 4 \times 4$. The workload of the MC task of a MB is then predicted as the sum of the six sub MC tasks.

3.4 The Total Workload

The total workload of a MB is modeled by summing the workload of VLD, IQ, DC-AC Prediction, IDCT, MC tasks plus a safety margin, which is a constant parameter.

All the parameters of the model depend on the run-time platform and decoder implementation. For a particular platform and decoder implementation, the parameters can be obtained offline. Using our model, the processing time required for workload prediction is 30 times or more faster than real time. Experimental results show that processing a thirty-second MPEG-4 video takes less than 1 second, on a PC with Pentium-4, 2.0GHz processor and 1 GB memory. The overhead involved is negligible, so this workload prediction model can be applied to real-time applications.

4. TRANSCODING SCHEME

This section presents our transcoding scheme that takes in a video bitstream and transcodes it to a target one such that its decoding workload is below the device's constraint, while keeping the distortion between the original and target video bitstream minimal.

As mentioned in the previous section, the decoding workload depends on MB types, the number of non-zero DCT coefficients, the position the last DCT coefficient, motion compensation modes and motion vectors. Any of these values can be modified to decrease the decoding workload in order to satisfy the workload constraint. Modifying MB type, motion compensation modes or motion vectors requires the transcoder to transcode the original video bitstream in the cascaded way, i.e., the transcoder fully decodes the video and then re-encodes it. This is very time consuming. Our transcoding scheme is designed to operate in the compressed domain, i.e., the transcoder reduces the decoding workload by discarding the Huffman codes or drop frames. The advantages of such a design are two-fold. Firstly, the transcoder's computational complexity is relatively low and no frame buffer is needed. Secondly, we do not modify the MB type, motion compensation mode or motion vectors during transcoding; therefore this known information can still be used to control the target workload.

The proposed transcoding scheme is done in the following three steps:

Workload Control: Given the constraint, the decoding workload is reduced by decreasing frame rate and dropping Huffman codes. This step may generate more than one candidates having the workload below the workload constraint.

Distortion Minimization: Among the candidates, the one with minimal distortion from the original video bitstream is chosen as the final result, and its metadata is generated as input for the third step. The metadata is the file indicating which frames and Huffman codes should be dropped. It is to be noted that the actual transcoding is done only in Step 3.

Actual Transcoding: the transcoder reads the metadata and performs the actual transcoding of the original video bitstream to the target video bitstream.

In the next subsections 4.1 and 4.2, more details about *Workload Control* and *Distortion Minimization* will be discussed respectively.

4.1 Workload Control

In this step, we reduce the decoding workload of the video bitstream by decreasing frame rate and dropping Huffman codes. The challenge is that the target frame rate is unknown and it is also not known how many Huffman codes should be dropped so that the target workload can be exactly below the device's constraint. Since the target frame rate must be below the original frame rate (which is normally 25 or 30 frame per second), the number of possible frame rates is limited. Therefore, all possible frame rates can be enumerated. For each frame rate, frames from the original video bitstream are dropped according to the frame rate. After that, using the proposed workload prediction model in Section 3, decisions are made as to which Huffman codes should be discarded for the remaining frames. The details are shown in Algorithm 1.

```
Input: Target Workload ($InputTarWL$)
Output: Metadata for candidate video bitstreams
GetOriInfo(); /* to get necessary information from the original video bitstream */
foreach FrameRate $fr$ do
    DropFrame($fr$);
    if ($MinReqWL(fr) \geq InputTarWL$) then
        continue
    end
    $TotalTarWL = InputTarWL$
    foreach remaining frame $f_{curr}$ do
        AllocFrameWL($f_{curr}$);
        DiscardHuffman($f_{curr}$);
        Update($TotalTarWL$);
    end
end
```

Algorithm 1: Workload Control

DropFrame(fr):

This procedure will specify which frame to be dropped to fit the frame rate. To ensure the remaining frames decodable, we first drop B-frames, then P-frames from the tail of every GOP and then I-frames. The frames are dropped evenly to avoid jittering.

AllocFrameWL(f_{curr}):

For the current frame rate fr, we denote the N frames which are kept after DropFrame(fr) as f_0, f_1, ..., f_{N-1}. Let f_{curr} be the current frame. The decoding tasks of each MB are divided into two parts: *Huff Comp* includes the tasks with workload depending on the number of Huffman codes; *Non-Huff Comp* includes the rest of the tasks with workloads unchanged after transcoding. We denote the original workload of *Huff Comp* of the remaining frames as *OriHuffWL[f_{curr}]*, *OriHuffWL[f_{curr+1}]*, ..., *OriHuffWL[f_{N-1}]*. The workloads of *Non-Huff Comp* of the remaining frames are denoted as *OriNonHuffWL[f_{curr}]*, *OriNonHuffWL[f_{curr+1}]*, ..., *OriNonHuffWL[f_{N-1}]*. The workload of these components were estimated in the function *GetOriInfo()* of Algorithm 1.

We denote $TotalTarHuffWL$ the total target workload of the *Huff Comp*. It is calculated as:

$$TotalTarHuffWL = TotalTarWL - \sum_{i=f_{curr}}^{f_{N-1}} OriNonHuffWL[i] \quad (5)$$

The target workload of *Huff Comp* for the current frame, $TarHuffWL[f_{curr}]$ can be calculated as:

$$TarHuffWL[f_{curr}] = OriHuffWL[f_{curr}] \times \frac{TotalTarHuffWL}{\sum_{i=f_{curr}}^{f_{N-1}} OriHuffWL[i]} \quad (6)$$

DiscardHuffman(f_{curr}):

The details of this function is shown in Algorithm 2.
In Algorithm 2, function *Discard(DCT_Pos)*, "the Huffman codes after *DCT_Pos*" are the Huffman codes with position after *DCT_Pos*, in zig-zag sequence. *DCT_Pos* is iterated from the 63 to 0 so that the less important Huffman codes are dropped first.

```
**Input**: $TarHuffWL[f_{curr}]$
**Output**: Metadata, assigned workload
**if** ($TarHuffWL[f_{curr}] \geq OriHuffWL[f_{curr}]$) **then**
    return $OriHuffWL[f_{curr}]$;
**end**
**for** $DCT\_Pos$ = 63 .. 0 **do**
    Discard($DCT\_Pos$); /* drop the Huffman codes after DCT_Pos of all the blocks in the current frame */
    Calc($Huff\_WL$); /* workload of the Huff Comp after discarding the Huffman codes */
    **if** ($Huff\_WL \leq TarHuffWL[f_{curr}]$) **then**
        return $Huff\_WL$;
    **end**
**end**
```

Algorithm 2: Discard the Huffman Codes

Update(TotalTarWL):

After discarding the Huffman codes, *TotalTarWL* is updated. Since the workload of *Non-Huff Comp* does not change, the *TotalTarWL* for the remaining frames is updated by:

$$TotalTarWL = TotalTarWL - OriNonHuffWL[f_{curr}] - TarHuffWL[f_{curr}] \quad (7)$$

4.2 Distortion Minimization

In the previous subsection, the transcoder generates all possible candidates that satisfy the device's workload constraint. In this step, CDDM is proposed to select the candidate with the best quality. It is done by estimating the distortion between the candidates and original video bitstream. The candidate with the least distortion is selected for transcoding.

The first issue is to choose a method to measure distortion between the original and the candidates. Traditionally, average PSNR or MSE are used to measure the video distortion. However, average PSNR and MSE can only measure spatial distortion but not temporal distortion [5]. In other words, the average PSNR or MSE cannot measure the difference between a video bitstream of 30fps and that of 5fps. However, the frame rate does affect the perceived visual quality. The studies of subjective video quality [4, 10, 20, 21] show that people perceive video quality degradation as the frame rate decreases. While the traditional average PSNR or MSE cannot measure the tradeoff between the frame rate and single frame quality, the works in [8, 15] provide a solution, replacing the dropped frames by the previous frames to compute the average PSNR. The reason is because player maintains the current frame on the screen before displaying the next frame. With this method, both spatial distortion and temporal distortion can be considered when using the average PSNR as the distortion measure.

The second issue is that the actual calculation for PSNR requires the original video bitstream and all the candidates to be decoded fully. This is very time-consuming, and unfeasible for transcoding on-the-fly. In our scheme, it is not necessary to have the exact PSNR to select the best candidate. A rough estimation of all candidates' distortions is sufficient.

Wu et al. [22] also considered the tradeoff between the spatial and temporal distortion. However, their approach is not suitable for our transcoding scheme. The main difference between their and our method is that the spatial quality is scaled with different quality levels in their method, while we simply drop the least important Huffman codes. Therefore, we cannot use the quantization value to calculate the spatial distortion. Furthermore, the relationship between the perceptual video distortion and frame rate should be linear (i.e., distortion between 5fps and 10fps should be more than distortion between 10 fps and 15 fps). However, the video quality in [22] is calculated by multiplying spatial distortion with the frame rate which may not be accurate.

In the rest of this section, a new algorithm is proposed to estimate the distortion between the original and transcoded video bitstreams. The algorithm is operated in the compressed domain to minimize transcoding overhead. The estimation algorithm uses information from the metadata, which makes the whole transcoding scheme even faster.

The proposed scheme reduces the decoding workload by dropping Huffman codes and frames. Dropping Huffman codes causes spatial distortion while dropping frames causes temporal distortion. To simplify the problem, we analyze the two distortion separately and then combine them. Before we go to the details of the algorithm, we first introduce some notations:

- $D(F_A, F_B)$ is the estimated distortion between frames F_A and F_B.
- $D_S(F_A, F_B)$ is the estimated spatial distortion between frames F_A and F_B.
- $D_T(F_A, F_B)$ is the estimated temporal distortion between frames F_A and F_B.
- $H(F)$ is the number of non-zero DCT coefficients of the frame F.

4.2.1 Spatial Distortion

Spatial distortion happens when Huffman codes are dropped during transcoding. Therefore spatial distortion is related to the number of Huffman codes dropped. For I-frames, the number of Huffman codes can be used directly to measure the spatial distortion. But, for P- and B-frames, distortion propagation has to be considered as well. It is because the frames that P- and B-frame depend on could also be changed. In our algorithm, the spatial distortions caused by dropping Huffman codes for different types of frames are estimated by the following equations:

For I-frame

$$D_S(I, I') = H(I) - H(I') \tag{8}$$

where I and I' are the original and transcoded frames.

For P-frame

$$D_S(P, P') = W \times D_S(F, F') + (H(P) - H(P')) \tag{9}$$

where P and P' are the original and transcoded frames; F and F' are the frames P and P' depend on, respectively; W is a parameter for presenting the attenuation effect for distortion propagation. In our algorithm, we set W to 0.5 based on experiments.

For B-frame

$$\begin{aligned} D_S(B, B') &= W \times (D_S(F_1, F_1') + D_S(F_2, F_2'))/2 \\ &\quad + (H(B_i) - H(B_i')) \end{aligned} \tag{10}$$

where B and B' are the original and transcoded frames; F_1, F_2 and F_1', F_2' are the frames B and B' depend on, respectively; W is the same parameter as in Equation 9.

4.2.2 Temporal Distortion

In addition to dropping Huffman codes, frames are also dropped during transcoding, resulting in temporal distortion. As mentioned before, the temporal distortion is estimated by replacing the dropped frame by its previous undropped frame. We calculate the distortion for every individual frame and sum the result up as the distortion for the whole video. We present how to estimate temporal distortion for different types of frames in the following paragraph. To simplify the problem, we assume the transcoder does not drop any Huffman coefficient.

For P-frame
Assume P_1 and P_2 are two P-frames in the original video and P_2 depends on P_1. After transcoding, P_1 is transcoded into P_1'. P_2 is dropped and is replaced by P_1'. Now we want to estimate the distortion between P_2 and P_1'. Since by assumption, the transcoder does not drop any Huffman coefficient from P_1, P_1 and P_1' are identical. The distortion between P_1' and P_2 should be equal to the difference between P_1 and P_2. Since P_2 depends on P_1, the difference between P_1 and P_2 can be estimated by the residual error after motion compensation. The residual error again can be estimated by the number of Huffman codes of P_2:

$$D_T(P_1, P_2) = H(P_2) \tag{11}$$

It is to be noted that a dropped P-frame may not be replaced by the frame it depends on. But it must be replaced by the frame in its dependency chain. So a more generic equation for estimating the distortion between a dropped P-frame and the replacing frame is:

$$\begin{aligned} D_T(P_0, P) &= W \times D_T(P_0, P_1) + D_T(P_1, P) \\ &= W \times D_T(P_0, P_1) + H(P) \end{aligned} \tag{12}$$

where P is the dropped P-frame, P_0 is the frame replacing P and P_1 is the frame P depends on. It is noted that P_0 and P_1 can be the same frame and they can be either P- or I-frame. W (the same parameter in Equation 9) is the parameter representing the attenuation effect for distortion propagation.

For B-frame
Estimating the distortion for a B-frame is more complex because B-frame depends on two frames and a dropped B-frame can be replaced by a frame that is not in its dependency chain. If a dropped B-frame is replaced by a frame that is in its dependency chain, we estimate the distortion by:

$$\begin{aligned} D_T(B, P_0) &= W \times (D_T(P_0, P_1) + D_T(P_0, P_2))/2 \\ &\quad + H(B) \end{aligned} \tag{13}$$

where B is the dropped B-frame, P_1 and P_2 are the frames B depends on. P_0 is the frame to replace B; and P_0, P_1 and P_2 can be the same frame and they can be either P- or I-frame. W (the same parameter in Equation 9) is the parameter representing the attenuation effect for distortion propagation.
If a dropped B-frame is replaced by a frame that is not in its dependency chain, the frame replacing it must be another B-frame having the same dependent frames as the dropped B-frame. We estimate the distortion by:

$$D_T(B, B_0) = H(B_0) + H(B) \tag{14}$$

where B is the dropped B-frame and B_0 is the frame replacing B.

For I-frame
In our scheme, we drop I-frame only after all the P- and B-frames are dropped. So the dropped I-frame must be replaced by another I-frame. We estimate the distortion by:

$$D_T(I, I_0) = H(I) + H(I_0) \tag{15}$$

where I is the dropped I-frame and I_0 is the frame replacing I.

4.2.3 Total Distortion

Now we combine spatial distortion and temporal distortion together. Assume F is the original frame. It is dropped during the transcoding. F'_0 is the frame replacing F and F_0 is the original frame of F'_0. We estimate the distortion between F and F'_0 by

$$D(F, F'_0) = W_S \times D_S(F_0, F'_0) + W_T \times D_T(F, F_0) \quad (16)$$

where W_S and W_T are the weight for spatial distortion and temporal distortion, respectively. It is difficult to select optimal values for W_S and W_T, because for different video content, the optimal value can be different. For example, when the motion of the video is low, the spatial distortion is more important, thus W_S should be larger than W_T, and vice versa. In our current implementation, considering the balance for all the cases, W_S and W_T are set to 0.5.

5. EVALUATION

The workload prediction model and the transcoding scheme are evaluated in this section.

5.1 Workload Prediction Model Evaluation

In this subsection, we evaluate the workload prediction model presented in Section 3. In the experiments, we considered two sets of video bitstreams: the training set and testing set. We measured the actual workload of the video bitstreams in the training set. Based on the actual workload and information extracted from the video bitstreams, we established the workload prediction model. Then, we used this model to predict the workload of the video bitstreams in the testing set. The prediction is in the Macroblock granularity level.

5.1.1 Experiment Setup

We ran the experiments on three different target platforms with three different decoders. The three target platforms are IBM X-31 laptop (600MHZ Pentium M processor, 256MB RAM, with Windows XP OS installed), SimpleScalar emulator [6] (sim-safe profile) and HP iPAQ hx4700 series PDA (624 MHZ Intel PXA270 processor, 64MB RAM, 128MB ROM, with Windows Mobile 2003 OS installed). The three decoders are the reference MPEG-2 decoder (TMN5) [2], the reference MPEG-4 decoder (MOMUSYS) [11] and an optimized MPEG-4 decoder in TCPMP project [1]. On SimpleScalar, we measured the number of instructions as decoding workload. On the IBM laptop and PDA, we measured execution time as decoding workload. In the experiments, we had 12 CIF raw videos with different contents shown in Table 1.

Each of the video content was encoded in MPEG-2 and MPEG-4 format with four bit rates: 256 KBps, 512 KBps, 768 KBps and 1024 KBps. In total, we had $4 \times 12 = 48$ videos encoded for MPEG-2 and MPEG-4 format respectively. The frame rate was 25 fps, and GOP size was 10. One or two B frames were inserted between two I/P frames. For each experiment, we divided the 48 encoded video bitstreams into 4 equal sets: set A, B, C and D. We re-ran each experiment 4 times. For each run, one set was picked as the testing set, and the remaining 3 sets are used as the training data. Using the training data, we run experiments to get the actual workload and the corresponding compressed domain information for the specific target platform, video format and decoder implementation. Using this information, the model's parameters for the platform, video format and decoder implementation are determined. Then the built model is used to predict the decoding workload of the 12 video bitstreams in testing set.

Table 1: 12 CIF raw videos

No	Video name	Description
1	akiyo	Still background and a foreground object with very low movements
2	bridgeclose	Still background and some small objects with random movements
3	bridgefar	Almost a still image
4	coastguard	Still background and two foreground objects with contrary movements
5	container	Still background and two foreground objects with same movements
6	foreman	Background and foreground have moderate movements
7	hall	Still background and two objects with moderate movements
8	highway	Background with very fast movements
9	mother-daughter	Still background and two objects with very slow movements
10	news	Still background, an object with fast movements and two objects with very low movements
11	silent	Still background and an object with moderate movements
12	walk	Both background and two foreground objects are with very fast movements

5.1.2 Results and Analysis

Figure 2 shows the experimental results of the workload prediction model. X-axis represents the prediction error rate, which is calculated by:

$$error\_rate = \frac{|actual\_workload - predicted\_workload|}{actual\_workload} \quad (17)$$

The Y-axis represents the percentage of MBs that were predicted below an error rate in X-axis. The three curves in each graph indicate the prediction result for the reference MPEG-2 decoder, reference MPEG-4 decoder and TCPMP MPEG-4 decoder.

In Figure 2, graphs *Laptop (1st run)* and *Laptop (3rd run)* show the results on the IBM laptop in the first and the third run. Graphs *SimpleScalar (1st run)* and *SimpleScalar (3rd run)* show the results on the SimpleSalar; and graphs *PDA (1st run)* and *PDA (3rd run)* show the results on the PDA. We do not show the result of the second and the forth run, due to limited space, but the results of the other two runs are very similar. This implies that the model is not biased towards any particular video bitstream.

The results show that on both laptop and SimpleScalar, for both MPEG-2 and MPEG-4 reference decoder, more than 90% of MBs were predicted below an error rate of 10% and 98% of MBs were predicted below an error rate of 20%. But on the PDA, only 40% of MBs were predicted below

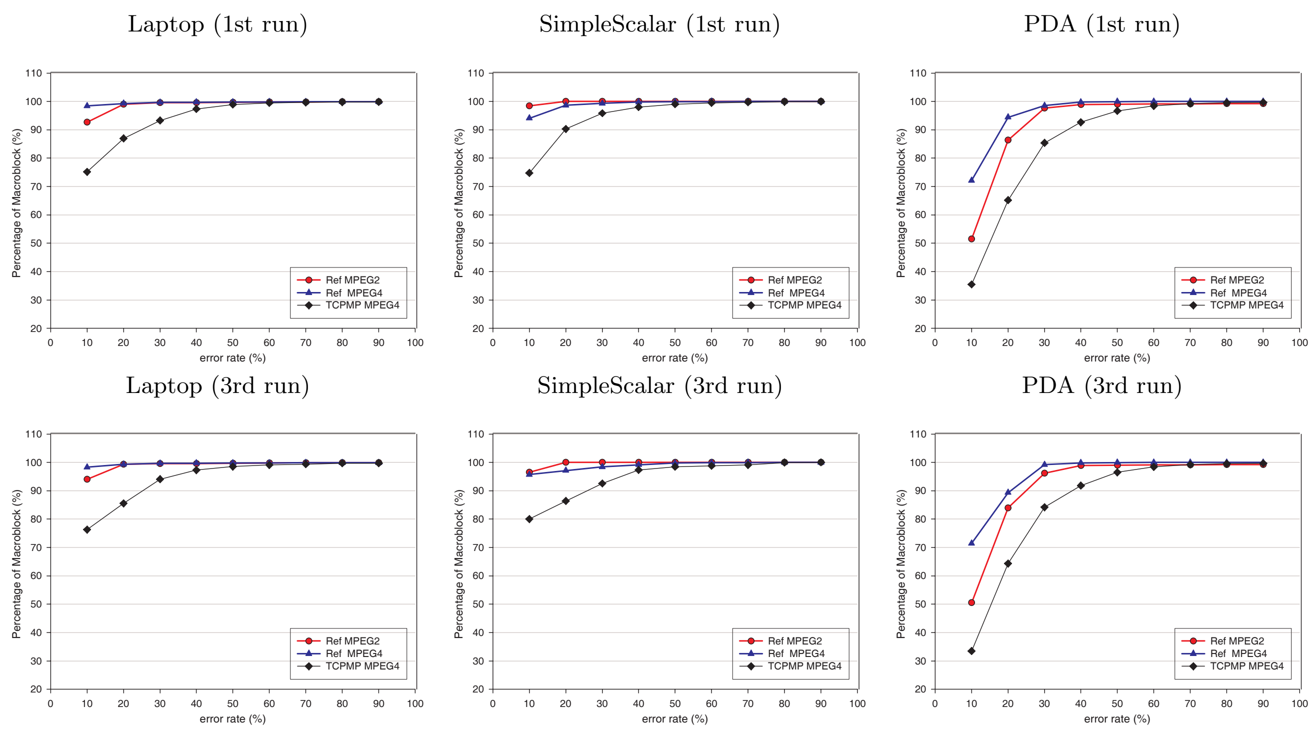

Figure 2: Cumulative prediction error rate of the workload prediction model

an error rate of 10% and 90% of MBs were below an error rate of 20%. Compared to the results on the laptop and SimpleScalar, the prediction on PDA is much less accurate. Our analysis showed that the error mainly came from the MC task. It was because the MC task has to perform many memory access operations and execution time for one memory access on PDA varied significantly in cases of cache hits and cache misses. On PDA, the cache size was small, which caused many cache misses. This in turn made the execution time less predictable. Since the PDA did not provide any mechanism for us to obtain the number of instructions, we could only use the execution time as the measurement for workload. This was why our model did not perform well on the PDA. On the laptop, the cache size was large, cache misses did not happen frequently. The execution time was not affected by the cache very much, so the model performed better. On SimpleScalar, we directly measured the number of instructions, which was not affected by the cache misses at all. That is why the prediction on SimpleScalar was the most accurate one.

The results also show that the prediction on the TCPMP MPEG-4 decoder was worse than on the other two decoders. For TCPMP MPEG-4 decoder, the percentage of MBs that were predicted below an error rate of 10% is about 20% less than the percentage for the other two decoders. It is because the TCPMP MPEG-4 decoder has a very optimized design. Its implementation has many branches that are not related to the bitstream content. The information of the bitstream content is not enough to predict these branches.

Figure 3 shows the comparison between our model and the history-based model proposed in [9]. The experiments were run on the laptop using TCPMP MPEG-4 decoder. The history-based model predicts the workload of the current frame by the weighted-average of previous same-type frames' workload. In the experiments, we set the size of the history window to 5 and the weight of each frame in the window to 0.2. The three curves show the prediction result of our model, the history-based model and the actual workload in frame sequence. It is observed that the curve of the proposed model matches the curve of the actual workload much better than the history-based model. The correlation coefficient between the history-based model and the actual workload was 0.54 and the average error rate was larger than 20%. However, the correlation coefficient between our model and actual workload was 0.91 and the average error rate was less than 2%. This shows the advantage of our model.

5.2 Transcoding Scheme Evaluation

In this experiment, using the scheme proposed in Section 4, we transcoded existing video bitstreams for different decoding workload constraints. The transcoding scheme is evaluated with the following two aspects: 1) whether *Workload Control* algorithm can accurately control the target decoding workload, which should be just below the device's constraint; 2) whether *CDDM* is accurate enough. That is whether the estimation result matches that of the traditional PSNR result.

To evaluate the first aspect, we measured the actual decoding workload of the target video bitstream and compared it to the original workload constraint. To evaluate the second aspect, we decoded all the candidates generated in Step 1. Then we calculated the actual PSNR between the candidates with the original video bitstream. We checked if the result we selected in Step 2 had a highest PSNR (the higher the average PSNR is, the less distorted the transcoded video bitstream is compared to the original video bitstream).

5.2.1 *Experiment Configuration*

The same 48 video bitstreams described in Section 5.1 were used in this evaluation. We ran experiments for 8 different workload constraints corresponding to the processor frequencies (instructions/sec) of 100, 150, 200, 250, 300, 350, 400 and 500 MHz. Assuming the original frame rate to be 25 fps, the possible target frame rates were set to be 5, 8, 10, 15, 18, 20, 22 or 25 fps. The actual workload was measured on SimpleScalar.

5.2.2 *Workload Control*

The Figure 4 shows the comparison between the actual decoding workload of our transcoded bitstream and the constraint. The X-axis represents the processor frequencies and the Y-axis represents the workload. Figure 4 shows how accurately our transcoding scheme could control the workload of the transcoded video bitstreams. The curve labelled *Workload Constraint* represents the constraints. The curves labelled 256KBps_Actual, 512-KBps_Actual, 768KBps_Actual, and 1024KBps_Actual represent the average of the actual workload of the video bitstreams with original bit rate of 256KBps, 512 KBps, 768 KBps and 1024 KBps, respectively. It is observed that all the 4 curves are all below and close to *Workload Constraint* curve showing that the workloads of all transcoded video bitstreams are kept under the workload constraint. Another observation is that the difference between the actual workload and the constraint was large when the processor frequency was 500MHz. This was because that processor frequency of 500MHz is more than enough to decode the original video bitstreams.

5.2.3 *Compressed Domain Distortion Measure (CDDM)*

After Step 1 of the transcoding scheme, we had the metadata of all possible candidates. In this experiment, we performed the transcoding of all these video bitstreams and calculated the actual PSNR from the original video bitstream. Using CDDM, we estimated the distortion of these transcoded video bitstreams from the original, and then we compared the estimated distortion values with the (1/Actual PSNR) values.

Figure 5 shows a comparison between the CDDM value and the corresponding (1/Actual PSNR) value for video *"news"* with bit rate of 512 KBps and processor frequency of 500 MHz. The matching of the 2 curves implies a high correlation between the CDDM and actual PSNR. Figure 5 shows that our *CDDM* correctly estimated the distortion for this test run of video *"news"* with bit rate of 512 KBps and processor frequency of 500 MHz. In total, we have conducted 192 such test runs. For one test run, if our algorithm selected the candidate with lowest (1/Actual PSNR) value, the selection was correct; otherwise, the selection was wrong. Figure 6 shows the accuracy of the *CDDM* for different processor frequencies. On average, in more than 92% of 192 experiments, our estimation algorithm selected the best quality video bitstreams, and in the rest 8%, the second best quality video bitstreams were selected.

6. CONCLUSION AND FUTURE WORK

We have presented a general model for predicting decoder workload of MPEG video. We verify the predictive power of this model by comparing it with well-known existing methods and actual workload measured on the device. We found

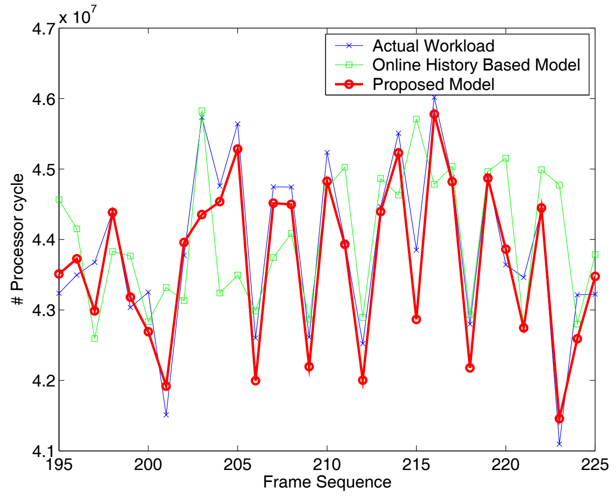

Figure 3: The comparison between proposed model and online history-based model

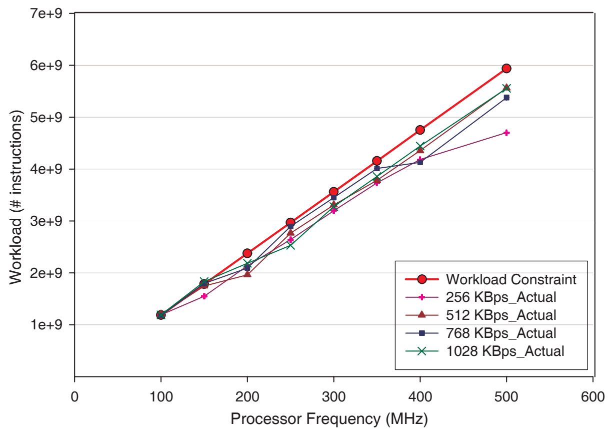

Figure 4: The comparison for the actual decoding workload and workload constraint

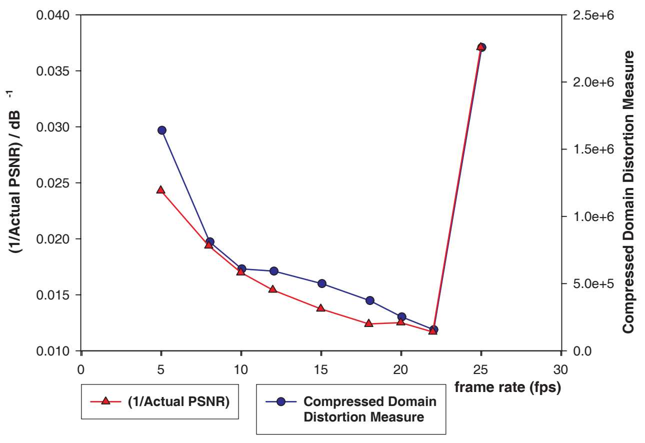

Figure 5: Comparison between the Compressed Domain Distortion Measure and 1/Actual PSNR

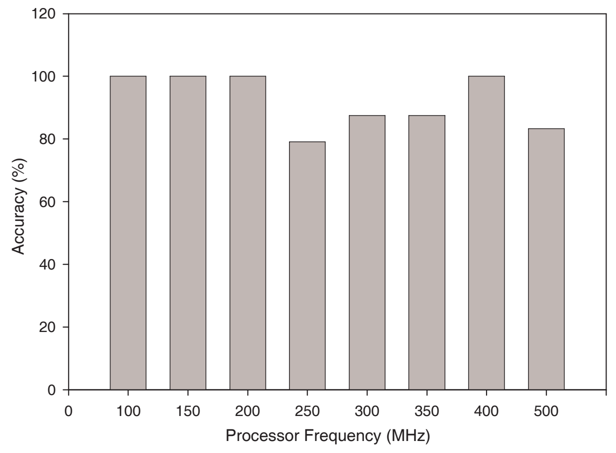

Figure 6: Accuracy of the Compressed Domain Distortion Measure

that in the frame granularity, the average prediction error between the model and the actual workload was less than 2% with different video formats and decoder implementations. We believe the value of our model is in providing a basis for guiding low complexity embedded system design and many other relevant tasks.

We then demonstrated how the model can be used in a MPEG video transcoding scheme, which is designed to provide an optimal match between the transcoded bitstream and a mobile device's processing power. In our transcoding scheme, we have further proposed two novel methods, namely, a workload control method and a compressed domain distortion estimation method. Both methods have been evaluated with experiments and were shown to be effective. The main advantage of our compressed domain transcoding scheme is its speed. Unfortunately, this is accompanied by an inherent disadvantage of inflexibility such as the inability of spatial scalability. This problem will be addressed in our future work.

7. REFERENCES

[1] http://tcpmp.corecodec.org/.

[2] http://www.mpeg.org/mpeg/mssg/.

[3] I. Ahmad, X. Wei, Y. Sun, and Y. Zhang. Video Transcoding: An Overview of Various Techniques and Research Issues. *IEEE Trans. on Multimedia*, pages 793–804, Oct 2005.

[4] A.H. Anderson, L. Smallwood, R. MacDonald, J. Mullin, and A. Fleming. Video data and video links in mediated communication: What do users value. *International Journal of Human Computer Studies*, pages 165–187, 2000.

[5] R. Apteker, J.A. Fisher, V.S. Kisimov, and H. Neishlos. Acceptability and Frame Rate. *IEEE Trans. On Multimedia*, pages 32–40, 1995.

[6] T. Austin, E. Larson, and D. Ernst. (SimpleScalar): An infrastructure for Computer System Modeling. *IEEE Computer*, pages 59–67, 2002.

[7] A. Bavier, B. Montz, and L. Peterson. Predicting MPEG Execution Times. *ACM SIGMETRICS Performance Evaluation Review*, 1998.

[8] M. Bonuccelli, F. Lonetti, and F. Martelli. Temporal Transcoding for Mobile Video Communication. *The second Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services*, 2005.

[9] K. Choi, K. Dantu, W. Cheng, and M. Pedram. Frame-Based Dynamic Voltage and Frequency Scaling for a MPEG Decoder. *ICCAD*, Nov 2002.

[10] G. Hauske, T. Stockhammer, and R. Hofmaier. Subjective Image Quality of Low-Rate and Low-Resolution Video Sequences. *Proc. Interational Workshop on Mobile Multimedia Communication*, Oct 2005.

[11] G. Heising and M. Wollborn. MPEG-4 version 2 video reference software package, ACTS AC098 mobile multimedia system (MOMUSYS). *IEEE Trans. on Consumer Electronics*, Dec 1998.

[12] T. Lan, Y. Chen, and Z. Zhong. MPEG2 decoding complexity regulation for a media processor. *IEEE MMSP*, 2001.

[13] Y. Liu, W. Lu, L. Wang, and K. Liu. Design of MPEG-2 to H.264/AVC Transcoder. *IEEE Tenth International Symposium on Consumer Electronics*, pages 1–3, 2006.

[14] M. Mattavelli and S. Brunetton. Implementing Real-Time Video Decoding on Multimedia Processors by Complexity Prediction Techniques. *IEEE Trans. on Consumer Electronics*, pages 760–767, Aug 1998.

[15] K. Ngan, T. Meier, and Z. Cheng. Imporved Single-Video Object Rate Control for MPEG-4. *IEEE CSVT*, pages 760–767, May 2003.

[16] W. Pan and A. Ortega. Complexity-scalable Transform Coding using Variable Complexity Algorithm. *Data Compression Conference*, Mar 2000.

[17] V. Patil, R. Kumar, J. Mukherjee, and S. Prasad. A Fast Arbitrary Down-Sizing Algorithm for Video Transcoding. *IEEE International Conference on Image Processing*, pages 857–860, Oct 2006.

[18] M. Schaar and Y. Andreopoulos. Rate-Distortion-Complexity Modeling for Network and Receiver Aware Adaptation. *IEEE Trans. On Multimedia*, pages 471–479, Jun 2005.

[19] D. Son, C. Yu, and H. Kim. Dynamic Voltage Scaling on MPEG Decoding. *ICPADS*, pages 633–640, 2001.

[20] G. Wilson and M.A. Sasse. Do users always know What's Good for Them? Utilizing Physiological Responses to Assess Media Quality. *Proceeding of HCI*, pages 151–175, Sep 2000.

[21] G.M. Wilson. Psychophysiological indicators of the Impact of Media Quality on Users. *Proceeding on HCI*, pages 95–96, Mar 2001.

[22] H. Wu, M. Claypool, and R. Kinicki. On Combining Temporal Scaling and Quality Scaling for Streaming MPEG. *Proceeding of NOSSDAV*, May 2006.

[23] C. Zhang, S. Zheng, C. Yuan, and F. Wang. A Novel Low-complexity and High-performance Frame-skipping Transcoder in DCT Domain. *IEEE Transactions on Consumer Electronics*, pages 1306–1312, Nov 2006.