Workload Model for Video Decoding and Its Applications

Huang Yicheng

Submitted in partial fulfillment of the

requirements for the degree

of Doctor of Philosophy

in the School of Computing

NATIONAL UNIVERSITY OF SINGAPORE

2008

Acknowledgments

I would like to take this opportunity to express my sincere thanks to many people, without whom this dissertation would not have been possible.

My foremost thanks go to my supervisor, Assistant Professor Wang Ye, who has had great impact on me. Over the past four years, he has set a good example for me to have great passion and a serious attitude about research. He has helped me overcome my shortcoming, set achievable objectives at each step, and kindled aspiration in my heart. Without him, this thesis would never have been completed. My gratitude also goes to Assistant Professor Ooi Wei Tsang and Assistant Professor Chan Mun Choon, who are members of my evaluation committee. They have provided me with valuable feedback to refine my research work. I would like to thank many friends in National University of Singapore for the inspiring discussions that have contributed to my research work and the many enjoyable hours we spent together for the leisure time. They are Tran Vu An, Huang Wendong, Hong Guangming, Zhu Zhehui, Zhang Bingjun, Gu Yan, Ni Yuan, Yu Jie, Liu Chengliang and Guo Shuqiao. I have really enjoyed the collaborations and discussion with these brilliant people.

Finally, I feel deeply indebted to my family members. Even though they know nothing about my research topic, they have listened to my explanation of the topic and encouraged me to pursue my dream. There are no words to thank them for that.

Contents

List of Figures

List of Tables

Abstract

.

In recent years, multimedia applications on mobile devices have become increasing popular. However, to design a mobile video application is still challenging due to the constraint of energy consumption. According to previous studies, the energy consumption of the mobile processor is cubic to its workload. For a mobile video application, it is therefore desirable to control decoding workload so that energy consumption by the processor may be reduced.

In this thesis, we study the relationship between decoding workload and video quality. Based on the analysis of video structure and decoder implementations, we propose a decoding workload model. Given a video clip, the model can accurately estimate the decoding workload on the target platform with very low computational complexity. Experiments are conducted to test the robustness of the model. The experiment results show that the model is generic to different decoder implementations and target platforms.

We also propose two relevant video applications: the decoding workload scalable transcoder and the decoding workload scalable encoder. Based on the decoding workload model, the proposed transcoder / encoder is able to generate a video clip which matches the decoding workload of the client while striving to achieve the best video quality. The transcoder /encoder can also balance the tradeoff between frame rate and individual frame quality, i.e., given a workload constraint, the transcoder / encoder can determine the most suitable frame rate /and individual frame quality combination even before the

actual transcoding / encoding. We achieve this by proposing two novel compressed domain video quality measures.

To my parents

Chapter 1

Introduction

1.1 Background

After a decade of explosive growth, mobile devices today are increasingly becoming important entertainment platforms for video and multimedia content. This application scenario is a fast emerging area with huge economic impact. However, supporting multimedia applications on mobile devices is more challenging due to constraints and heterogeneities such as limited battery power, limited processing power, limited bandwidth, random time-varying fading effect, different protocols and standards, and stringent quality of service (QoS) requirements.

Energy consumption is a critical constraint for a mobile video application. For years, chip makers have focused on making faster processors. Following Moore's Law, the processor's processing power would double every two years. However, the development of the battery has not improved as fast as that of the processor. As Figure 1.1 [68], CPU speed double per 18 months while battery energy density doubles per 12 years.

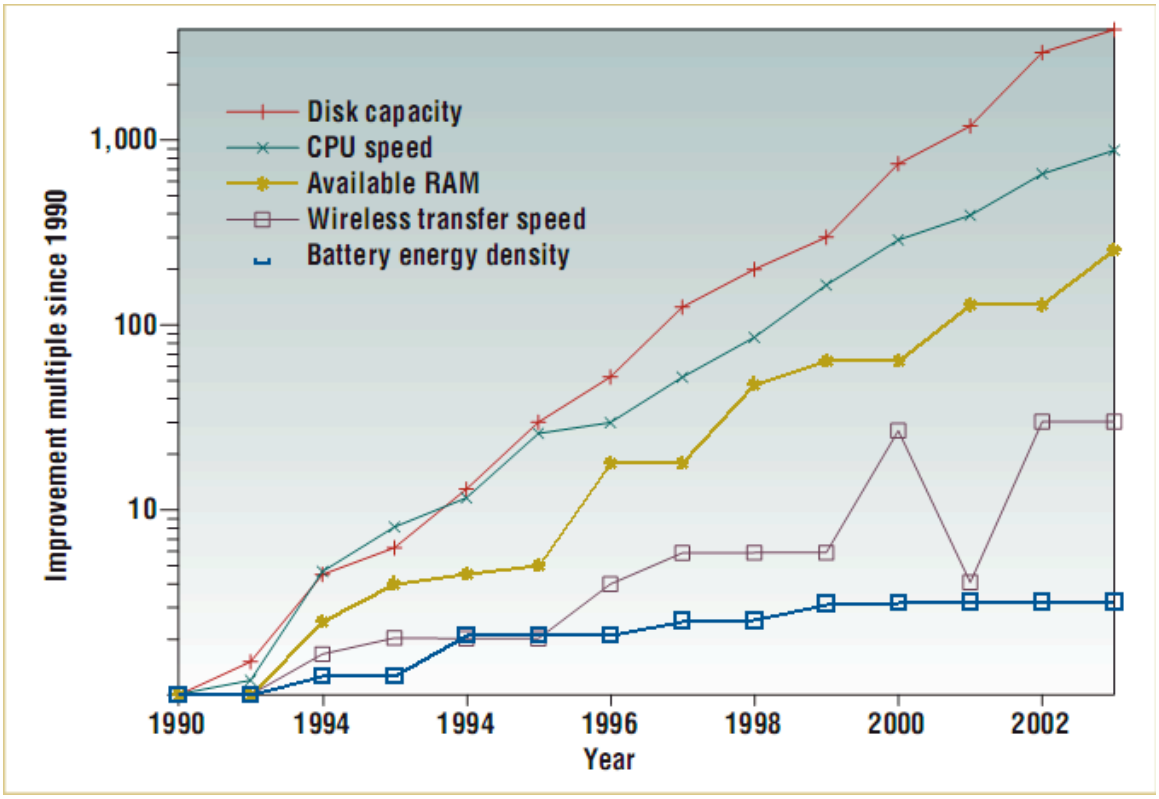

Figure 1.1 Improvement since 1990 (quoted from [68])

The battery of a typical mobile device such as a PDA or a mobile phone can only support video playback for about four hours. With streaming, battery lifespan will be even shorter as receiving data from a network requires substantial power. As a result, a mobile device has to minimize its energy consumption to prolong its battery life and attain suitable levels of quality of service at the same time.

Energy saving can be done at three levels in the computer system hierarchy: hardware, operating system and application. Energy at hardware level saving is out of the scope of this thesis. The advantage of saving energy at the operating system level is that the operating system has knowledge of the whole machine status, and so it can manage

energy consumption efficiently. This is why most energy saving schemes are done at this level [46, 47]. However, the operating system functions at a low level in the computer system hierarchy, and it therefore does not have knowledge of applications or users' behavior. This renders energy saving schemes at the operating system level incapable of adapting to different application scenarios or users' preferences. On the contrary, energy saving schemes at the application level know about the applications and users' behaviors, and are therefore able to make tradeoff between quality of service and energy consumption. For example, in a mobile video application, when energy is plentiful, application behavior should be biased toward good user experience: displaying video at a high frame rate / resolution; when energy is scarce, the behavior should be biased toward energy conservation: displaying video at a low frame rate /resolution. The problem is: how low should the frame rate / resolution be? On one hand, we know energy can be saved by sacrificing quality of service; on the other hand, we do not want to compromise too much on quality – the quality should still be acceptable. Ideally, therefore, quality should be optimized based on the available resources. From this aspect, a mobile video application design can be regarded as an optimization problem under multiple constraints. To solve such a problem, mathematical models between video quality and constraints should be established. For example, for the constraint of bandwidth, rate-distortion (R-D) models have been studied for decades. However, the current state of the energy-distortion model is far from satisfactory.

In a mobile device, energy is mainly consumed by three components: wireless network interface (WNIC), liquid crystal display (LCD) and processor. For WNIC, energy

consumption depends on whether the component is in active mode. Network reshaping schemes have been proposed to make WNIC remain in sleeping mode for as long as possible [43, 44, 45]. For LCD, it requires two power sources, a DC-AC inverter to power the cold cathode fluorescent lamp (CCFL) used as backlight, a DC-DC converter to boost and drive the rows and columns of the LCD panel. Energy is also consumed in the bus interface, LCD controller circuit, RAM array, etc. [48]. Energy consumption can be reduced by variable duty-ratio refresh, dynamic color depth control, and brightness and contrast shift with backlight luminance dimming [49, 50, 51, 52, 53]. The processor, which is a digital static CMOS circuit, can be calculated by Equation (1.1):

$$E \propto a_{0\to 1} * C_L * V_{dd}^2 * F_{CLK} \tag{1.1}$$

where F_{CLK} denotes clock rate (processor frequency), V_{dd} is supply voltage, C_L denotes node capacitance, and $a_{0\to 1}$ is defined as the average number of times in each clock cycle that a node will make a power consumption transition (0 to 1) [29]. The relationship between voltage and processor frequency follows Equation (1.2), based on the alpha-power delay model [30]:

$$\frac{1}{F_{CLK}} \propto \frac{V_{dd}}{(V_{dd} - V_T)^{\alpha}} \tag{1.2}$$

where V_T is the threshold voltage of the processor, and α is the velocity saturation index. From the above equations, we can calculate the energy consumption of the processor by

processor frequency, which can be regarded as the decoding workload for the mobile video application. Energy consumption can be reduced by adopting dynamic voltage scaling (DVS) schemes [54] or directly reducing workload.

As energy consumption of the processor can be derived from the decoding workload, we thus focus on the model between decoding workload and video quality and its relevant applications in this thesis. The study of the decoding workload model is important because: 1) As we have mentioned previously, a mathematical model can help us save energy as much as possible while still provide the quality of service which users prefer. 2) The model will still apply even if we adopt some operating system level energy saving scheme, for example DVS. The basic idea of DVS is to scale processor frequency as low as possible based on workload prediction. Energy can therefore be saved as energy consumption can be calculated by the processor frequency. However, workload prediction needs to be accurate. If the actual workload is more than the prediction, the video cannot be fully decoded, which results in bad quality; if the actual workload is less than the prediction, the frequency will be scaled too high, which results in a waste of energy. The model studied in this thesis is able to predict decoding workload accurately, thereby improving the performance of DVS schemes. 3) Decoding workload itself can also be a constraint: most existing mobile devices' processor frequencies are in the range of 200 MHz to 600MHz. It is difficult for them to decode a video clip encoded by complex codec technologies such as MPEG-4 and H.264 at a high frame rate (25 – 30fps). For such cases, our study can help to generate a video clip which meets the constraint of devices' processing power while still guarantees quality of service.

1.2 Challenges

In studying decoding workload and the relevant video applications, we face three major challenges:

First, we need to study the relationship between video bitstream and decoding workload. This is analogous to rate-distortion studies [56, 57, 58, 59, 60], which have found out that bit rate can be controlled by quantization scale. For decoding workload, we should find out similar key parameters and establish a mathematical model so that we can control the decoding workload by adjusting the parameters. The problem is that most existing video codecs are designed for the rate control. We can establish a model based on the current video codec's architecture or propose a new video codec specific to decoding workload control. In our opinion, designing a new video codec cannot be a practical solution especially when the new codec is not compatible with existing systems. Hence, in the thesis, we propose a decoding workload model for existing MPEG video formats and codecs. The model should be sufficiently accurate and fast. It should also be flexible enough so that it can be easily applied to different kinds of applications. Moreover, the model should be generic for adaptability to different video formats, decoder implementations and platforms.

Second, even with a decoding workload model, designing an application scheme remains difficult, e.g., to design a video encoder which generates a bitstream under the constraint of decoding workload. According to previous studies, different frames require different

amounts of decoding workload even under the same quality. In some extreme cases, the decoding workload of one frame can be 10 times different from that of another. If we allocate workload to frames evenly, quality will differ quite a lot. That results in unstable user experience. A better approach is to allocate workload based on requirements so that different frames may be of the same quality. That is why a sophisticated decoding workload control scheme is necessary. However, the scheme is difficult to design since the decoding workload requirement is affected by several factors: video content, encoding algorithm and video format. Taking all these factors into consideration makes the scheme very complex. Moreover, an objective measure for estimating the quality of the encoded frames or MBs is not available before the frames or MBs are actually encoded. This makes scheme design even more difficult.

Third, we need to consider the tradeoff between individual frame quality and frame rate. In traditional video applications, the frame rate is fixed at 25 or 30 frames per second, i.e., the decoder decodes a frame every 1/25 or 1/30 second. However, in mobile video applications, some mobile devices' processing power is so low that they cannot decode a normal quality frame properly within that time slot. Therefore, to fix the frame rate at 30 or 25 fps in the mobile application may not be feasible. To overcome the constraint, we can either reduce the frame rate or the quality of individual frames. The problem is, we may have more than one combination of frame rate and individual frame quality with the same decoding workload. To provide the best quality of service, we need to select the one with the best quality among them. Therefore, an objective measure is necessary to evaluate the quality of all the options.

1.3 Structure of Thesis

The rest of the thesis is organized as follows: A reader without knowledge about mobile video application design may want to refer to Chapter 2 for some background knowledge and related work, including that on MPEG video format, decoding workload model, existing energy saving schemes and objective video quality measures. In Chapter 3, we present our decoding workload model and evaluate it using different decoders on different target platforms. Based on the model, we propose two decoding workload related mobile video applications in Chapters 4 and 5. In Chapter 4, we propose a workload-scalable transcoder which works in the compression domain. It reduces the decoding workload by dropping either Huffman codes or frames. To evaluate the tradeoff between Huffman codes and frames, we propose mean compression domain error (MCDE), a compression domain video quality measure designed for transcoding applications. In Chapter 5, we propose a workload-scalable encoder. It includes two schemes: the frame rate selection scheme and the workload control scheme. The frame rate selection scheme selects the most suitable target frame rate before actual encoding; the workload control scheme controls decoding workload under the constraint. In Chapter 6, we conclude the thesis and present future directions.

1.4 Main Contributions

The major contributions of the thesis lie in three aspects:

First, we analyze the relationship between video quality and decoding workload, based on which we establish a mathematical decoding workload model. The experiments show that the model is accurate and fast. Moreover, it is generic to different video formats (with MPEG video structure), decoder implementations and target platforms.

Second, we study two decoding workload related video applications: transcoder and encoder. We study how to make them accurately control the decoding workload of the generated video bitstream while the quality of the video bitstream is optimal. We call this transcoder/encoder the decoding workload-scalable transcoder/encoder. To our best knowledge, this is the first attempt at studying decoding workload applications in such a comprehensive manner.

Third, we propose two compression domain objective video quality measures. Conventional video quality measures such as peak signal-to-noise ratio (PSNR) or mean square error (MSE) assume the frame rate is fixed. They only consider spatial distortion but not temporal distortion. The measures we propose in this thesis can take both spatial and temporal distortions into account. Furthermore, they can estimate the quality of the target video bitstream even before actual encoding or transcoding, with very low computational complexity. The measures can also help the transcoder and the encoder determine the target frame rate with very low complexity.

Chapter 2

Background and Related Work

2.1 Introduction

In this chapter we introduce the related works of this thesis. As the decoding workload model is established based on the video bitstream analysis, we first briefly introduce the MPEG video formats in Section 2.2. After that we survey the related works on the decoding workload model in Section 2.3. In Section 2.4, we introduce the existing energy saving schemes for the mobile video applications, which can be regarded as the background of the transcoder and encoder proposed in Chapters 4 and 5. In Section 2.5, we present the traditional objective video quality measures and show why they are not suitable for the mobile video applications. That is the reason why we propose new compression domain video quality measures in this thesis.

2.2 MPEG Video Format

In this thesis, our schemes are proposed mainly based on the MPEG video formats including MPEG-1 [69], MPEG-2 [70] and MPEG-4 [71]. Although they are different in the details, they share the similar bitstream structure and encoding/decoding procedure. An MPEG video sequence is made up of frames, which are of three different types: I-frame, P-frame and B-frame. Each frame consists of several slices, which again consist of Macroblocks (MBs). Encoding or decoding a video sequence can be regarded as encoding or decoding a sequence of MBs. An un-skipped MB can have three types: I-Type, P-Type and B-Type. An I-frame can only have I-Type MBs; a P-frame can have I- or P- type MBs and a B-frame can have all the three types of MBs.

To encode an I-Type MB, the data are first transformed from the spatial domain data to the discrete cosine transform (DCT) domain. The DCT domain data are known as DCT coefficients. After that, the DCT coefficients are quantized by the quantization scale, and then encoded into Huffman codes, which again encoded by the run-length coding into the target bitstream. To encode a P-Type MB, the encoder first finds out a most similar reference block in its previous I- or P-frame and calculates the difference, which is known as residual error, between the current MB and the reference block. This task is called motion estimation (ME). The residual error is then encoded by the same procedure as the I-Type MB. Encoding a B-Type MB is the same as with a P-Type MB except that the encoder finds two similar blocks from its previous and next I- or P-frame and uses their average to calculate the residual error.

The decoding procedure is an inverse to the encoding procedure: the decoder reads the run-length codes from the bitstream and decodes them to the Huffman codes. The Huffman codes are then decoded to the DCT coefficients. We call this task variance length decoding (VLD). After VLD, the DCT coefficients are inverse quantized (IQ) and then transformed into the spatial domain data by the inverse DCT (IDCT) task. If the MB is I-Type, the decoding procedure finishes after IDCT; if the MB is P- or B-Type, the spatial domain data get from IDCT task should be added with its reference block to form the final output. This task is called motion compensation (MC). Thus, the MBs in P- or B- frames are decoded dependent upon their reference block in its previous and next I- or P-frame. If its previous or next frame is not decoded correctly, the P- or B- frame cannot be decoded, either. In this case, we call the previous and next frames reference frames. A reference frame can also have its reference frame. These related frames form a chain, which is called dependent chain.

We note that although our research in this thesis is based on the MPEG video format, most of algorithms we proposed can also be applied to other video formats, such as H.261 [24] and H.263 [25], whose bitstream structures and encoding/decoding procedures are very similar with the MPEG video format. For the video formats which has extra encoding/decoding tasks, for example, H.264 [23] employs intra prediction sub-procedure for I-MB, we believe we can also extend our algorithm to adapt them in future work.

2.3 Decoding Workload Model

The existing decoding workload 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 [8] 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. [6] 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 [17] 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 (our experiments results shows that 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 decoding workload based on information extracted from the video bitstream. In [12], 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 the model does not need full video decoding. Prediction results can be inserted into the 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. [11] 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 [16] 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 [11, 12, 16] 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 [11, 12, 16] may not be generic for different decoder implementations and video formats. In the thesis, we propose a new decoding workload model. It estimates the decoding workload based on information of the video bitstream. The proposed model has advantages of being:

Accurate: Our experiments show that the model can estimate the decoding workload of a frame within an error rate of 2%.

Generic: The model applies to different video formats (with MPEG video structure), decoder implementations and target devices.

Fast: The model only needs the information from the compression domain for predicting, i.e. no IDCT or MC is needed during the runtime.

2.4 Energy Saving Schemes for Mobile Video Applications

For a mobile device, WNIC, LCD and processor are the three major parts consume the energy. The existing energy saving schemes may target on any one of them or all of them. As we focus on the processor component in this thesis, we only review the processor related schemes in the rest of this sub-section.

The schemes to save the processor energy for the mobile video applications work at three levels: hardware level, operation system level and application level. Hardware level is out of the scope of this thesis. Operation level schemes include two main directions: dynamic power manager (DPM) and dynamic voltage scaling (DVS). DPM-based techniques rely on switching off parts of a device (processor, memory, display, etc.) at runtime, based on their usage. On the other hand, DVS relies on changing the frequency or voltage of the processor at runtime to match the workload generated by an application.

DPM schemes have been studied in the works in [32, 33]. In [32], the approach is based on renewal theory. The model assumes that the decision to transition to low power state can be made in only one state. In [33], the model is developed based the Time-Indexed Semi-Markov Decision Process model (TISMDP). This model is complex, but also has wide applicability because it assumes that a decision to transition into a lower-power state can be made from any number of states.

The DVS approaches can be classified in two categories: feed forward and feed backward. Figure 2.1 outlines the general system architecture.

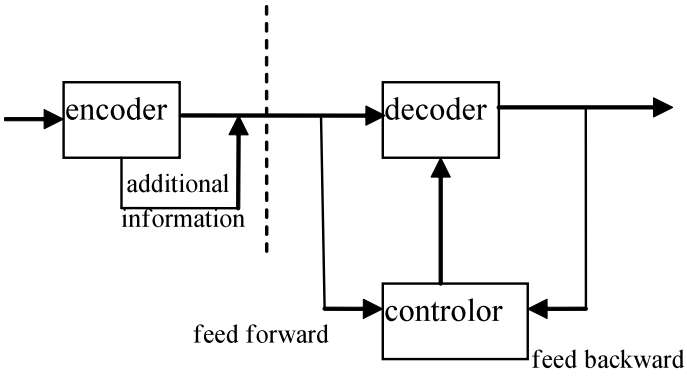

Figure 2.1 DVS system architecture

In a feed forward approach [34, 12] the encoder is modified to pass additional information about the decoding complexity as part of the frame header. This allows the controller at the decoder side to adjust the processor speed at the start of the decoding. In [34], the scheme stipulates the processor frequency range for every macroblock. The key idea is to make use of the input buffer and the playback buffer to adapt to the requirement variation. The frequency ranges at specific points in time are obtained by simulating a set of video streams. In [12], the proposed scheme divides the decoder into several parts and predicts each part by a linear equation. The parameters used by the linear equation are obtained by the simulation. The prediction does not need the actual decoding. The prediction results can be inserted into the frame header in any format.

In a feed backward approach the performance of the decoder is observed and subsequently adjusted. The most generic approach is to consider the decoder as a black box and observe its effect at the system level [31, 36, 37, 38, 39]. If the system information indicates that the decoder is running too fast, the processor frequency can be

reduced. The system information includes decoding time, the playback buffer and the processor utilization. Taking the decoder as an open box yields better results. In [40], the instruction latencies are classified as on-chip latencies and off-chip latencies. The on-chip latency is caused by events that occur inside the CPU. It may be reduced by increasing the processor clock frequency. The off-chip latency is independent of the internal clock frequency. The off-chip latency is able to be calculated via the record reported by performance-monitoring unit. The on-chip latency is predicted on the fly. The frame type is considered when calculating the off-chip latency. In [8], a frame-based DVS scheme is proposed. The scheme divides the decoding procedure into frame-dependent and frame-independent portions. Frame-dependent workload of the current frame is predicted by the weighted-average of previous same-Type frame's workload. The prediction error is compensated by scaling the processor frequency of frame-independent part. In [7], the scheme changes the processor frequency at the beginning of the GOP, which contains a number of frames. Two algorithms are proposed. The first algorithm scales the processor frequency according to the previous delay value. The second algorithm scales the frequency according to the previous workload as well as type and size of the incoming frames.

It is noted that the efficiency of the DVS schemes heavily relied on the workload prediction. As we mentioned in the previous sub-section, the existing workload model is not yet satisfied. The workload model we proposed in this thesis can be easily adopted in the existing DVS schemes and improve the performance.

At the application level, various schemes have been proposed: in the paper of [62, 64], the authors investigate the trade-offs between processing cost of less compression algorithms and networking. They suggest using different compression algorithm for different application scenarios. In [65], the authors propose an energy-optimized decoder implementation, which can reduce 10~12 percentage of the energy consumption on the ARM processor. Han et al proposed a transcoder between the original video source and the mobile device [63]. The transcoder reshapes the original video to reduce its decoding complexity. Jason et al propose a similar adaption scheme in [66]. However, the transcoding and adaptation schemes they propose can only resize the frame to one or two fixed sizes. They cannot adapt to the different workload constraints dynamically. That is exactly the advantage of the transcoder we propose in Chapter 4. In [67], He et al analyze the relationship among the power, rate and distortion for the video encoder applications. In Chapter 5, we propose a similar encoding scheme. The difference is He et al focus on the energy consumption of the encoder; we, on the other hand, focus on the decoder.

2.5 Objective Video Quality Measure

Conventionally, the video quality is measured by the sum of squared differences (SSD), mean squared error (MSE), peak signal-to-noise ration (PSNR) and the sum of absolute difference (SAD) [26], which calculate the distortion of every single frame by

$$SSD_{\mathrm{A}}(F,G) = \sum_{s \in A} |F(s) - G(s)|^2 \qquad (2.1)$$

$$MSE_A(F,G) = \frac{1}{|A|} SSD_A(F,G) \qquad (2.2)$$

$$PSNR_A(F,G) = 10\log\frac{(225)^2}{MSE_A(F,G)}db \tag{2.3}$$

$$SAD_A(F,G) = \sum_{s\in A} |F(s) - G(s)| \tag{2.4}$$

The distortion of the whole video sequence is then calculated as the mean of the individual frames, $D = \sum_{i=1}^{n} D(i)$, where *D(i)* is the distortion of the individual frame. These measures assume that the frame rate of the video sequence is fixed, which is exact in the case of the traditional video application. However, in the mobile video application, due to the limitation of the bandwidth or processing power, we may sacrifice the frame rate to improve the individual frame quality. In such a case, the conventional measures are not suitable [4]. It is because they only consider the spatial distortion caused by the lossy compression algorithm during the encoding. But they do not consider the temporal distortion caused by the un-continuous frame sampling.

A number of researchers have studied the perceptual video quality for low frame rate. In [18, 19], the authors measure the subjective video quality from the perception of physiological. The measured signals include Galvanic Skin Response (GSR), Heart Rate (HR) and Blood Volume Pulse (BVP). The results show that the physiological response to video degradation from 25fps to 5fps can be detected. Researches in [3] found that users do not subjectively detect the difference between 12fps and 10fps when engaged in a task. Although these work give out some findings and conclusions based on the

subjective testing, none of them can measure the quality of a given video sequence objectively.

In [27], the authors propose an objective measurement for low frame rate video by considering both spatial distortion and temporal distortion are considered. However, the approach is designed for their particular system rather than a generic video quality objective measurement. Moreover, their model is based on the generic rate-distortion theory, which is not accurate for the low bit rate video compression.

In [7, 28], the authors propose a measure for un-fixed frame rate video sequence using the traditional objective video quality measure such as MSE or PSNR. In practice, reducing the frame rate is implemented by dropping frames from the original frame sequence. At the client, the dropped frame can be considered as replaced by its previous frame in display order. The reason is because player maintains the current frame on the screen before displaying the next frame. The temporal distortion thus can be calculated as the distortion between the original frame and its replaced frame. The whole video sequence's distortion is calculated as the average PSNR/MSE of all the corresponding frames. Although this approach is good for measuring the quality of an existing video bitstream, it is too computationally expensive for those applications where the video bitstream does not exit. In the applications such as transcoder and encoder, we may have many candidate frame rates. We want to select out the best one before the actual transcoding or encoding. However, to calculate PSNR/MSE, this approach requests the

actual transcoding/ encoding and decoding for every candidate frame rate. This is very time-consuming and unfeasible to the real-time applications.

In this thesis, we propose two objective video quality measures in Chapter 4 and 5. They are designed for transcoding and encoding application, respectively. They can accurately estimate the target video quality for the un-fixed frame rate video sequences with very low computational complexity. We integrate the two measures into our workload-scalable transcoder and encoder to help to decide the best target frame rate before the actual transcoding and encoding.

Chapter 3

Decoding Workload Model

3.1 Video Decoding Procedure

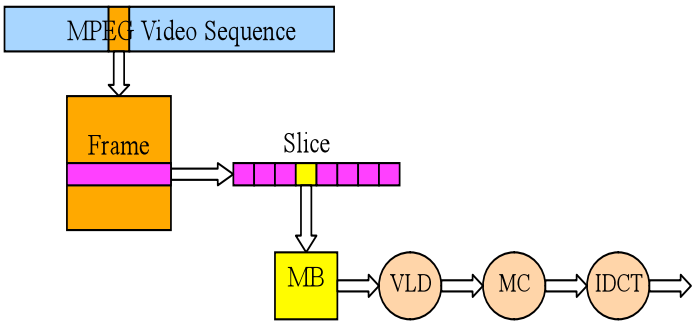

Figure 3.1 The decoding process of MPEG-2 video

In this section we present a new decoding workload prediction model to predict the decoding workload for MPEG video bitstream. As shown in *Figure 3.1*, a typical MPEG video bitstream 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.

3.2 Decoding Workload Model and Analysis

In this section, we model the decoding workload corresponding to the tasks VLD, IQ, DC-AC prediction, IDCT and MC for each MB. Our analysis is based the reference MPEG-2 decoder and reference MPEG-4 decoder. We run the decoders on SimpleScalar [5] instruction set simulator (with Sim-Profile configuration) and measure the processor cycles as the decoding workload. Since we envisage the decoder running on a general-purpose processor, we choose our processor to be a RISC processor (similar to a MIPS3000) without any MPEG-specific instructions. It is noted that, in practice, a video bitstream can be decoded by different decoders on different target platforms. The model should be designed to be generic to these decoders and platforms.

3.2.1 VLD, IQ and DC-AC Prediction Tasks

3.2.1.1 VLD Task

In MPEG video codecs, the 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.

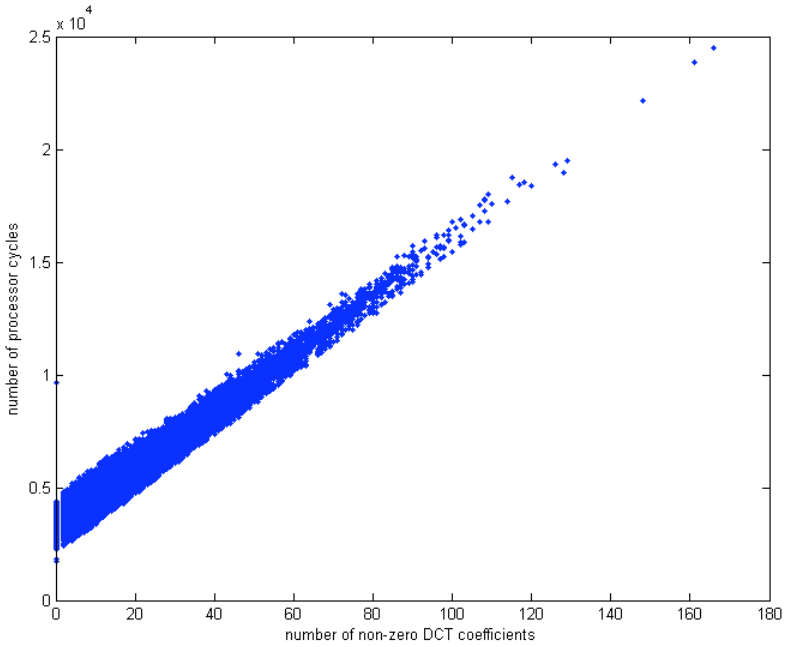

Figure 3.2 Workload generated by VLD task of the reference MPEG-2 decoder

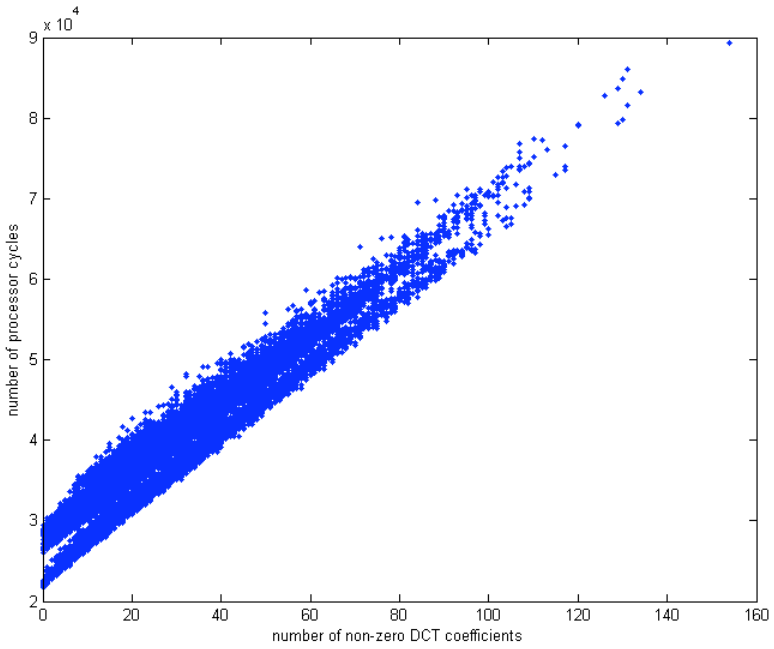

Figure 3.3 Workload generated by VLD task of the MPEG-4 decoder

Figure 3.2 and *Figure 3.3* show typical plots of the number of processor cycles required by the reference MPEG-2 decoder and MPEG-4 decoder's VLD task for different number of non-zero DCT coefficients in a MB. It is observed that both the plots form linear bands. Thus, we model the VLD task by $W_{vld} = a_{vld} \times n_{coef} + b_{vld}$, 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. And considering some decoder may implement VLD for Intra, Inter and Skipped MB differently for optimization, we get a more generic model for the VLD task:

$$W_{vld} = \begin{cases} a_{vld\_intra} * n_{ceof} + b_{vld\_intra} & Intra\ MB\ type \\ a_{vld\_inter} * n_{ceof} + b_{vld\_inter} & Inter\ MB\ type \\ b_{vld\_skip} & Skipped\ MB\ typ \end{cases} \qquad (3.1)$$

3.2.1.2 IQ Task

There are usually 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:

$$W_{iq} = a_{iq} * n_{ceof} + b_{iq} \qquad (3.2)$$

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.

3.2.1.3 DC-AC Prediction Task

The DC-AC Prediction task in MPEG-4 decoder is to estimate the DC or AC coefficients from the previous decoded DC and AC coefficients.

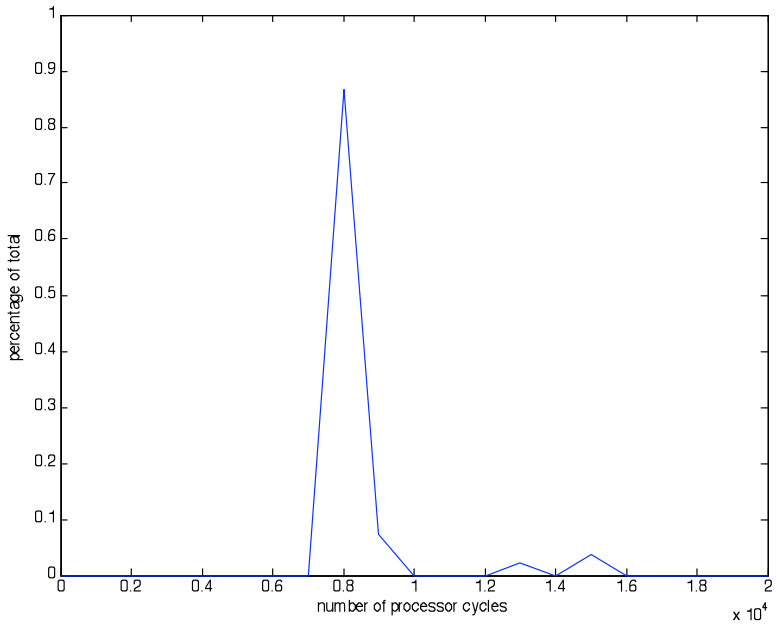

Figure 3.4 Processor cycles distribution of the DC-AC Prediction task of reference MPEG-4 decoder

Figure 3.4 shows a typical processor cycle distribution of the DC-AC Prediction task of the reference MPEG-4 decoder (MPEG-2 decoder does not have DC-AC Prediction task). It is observed that 90% MBs' DC-AC Prediction tasks cost a similar number of processor cycles. Hence, it is reasonable to approximate the DC-AC prediction task as a constant value. And again, considering that the decoder may have different DC-AC prediction implementations for different types of MBs for optimization, we model the DC-AC Prediction task by:

$$W_{dcac} = \begin{cases} b_{dcac\_intra} & Intra\ MB\ type \\ b_{dcac\_inter} & Inter\ MB\ type \\ b_{dcac\_skip} & Skipped\ MB\ typ \end{cases} \qquad (3.3)$$

where b_{dcac_intra}, b_{dcac_inter} and b_{dcac_skip} are parameters.

For MPEG-2 decoder, which does not have DC-AC Prediction task, this model can adapt by setting b_{dcac_intra}, b_{dcac_inter} and b_{dcac_skip} to zero.

3.2.1.4 VLD+IQ+DC-AC Prediction Tasks

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:

$$W_{vld} + W_{iq} + W_{dcac} = \begin{cases} a_{intra} * n_{ceof} + b_{intra} & Intra\ MB\ type \\ a_{inter} * n_{ceof} + b_{inter} & Inter\ MB\ type \\ b_{skip} & Skipped\ MB\ typ \end{cases} \quad (3.4)$$

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

3.2.2 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. We confirm this by the experimental results shown in *Figure 3.5*

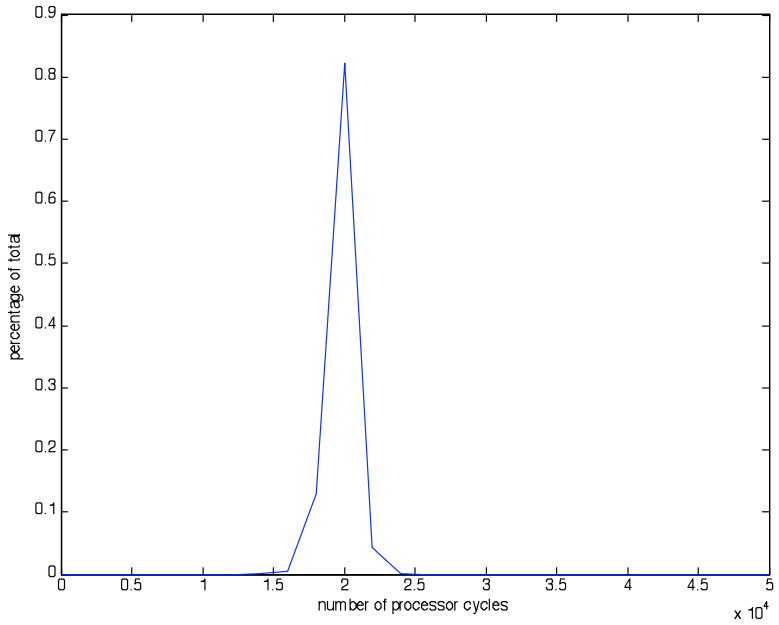

Figure 3.5 Processor cycles distribution of the IDCT task of reference MPEG-2 decoder

Figure 3.5 is the processor cycle distribution of the IDCT task of the reference MPEG-2 decoder. It is observed that most MBs' IDCT tasks cost the same number of processor cycles. However, some decoder implementation may optimize the IDCT task by considering the position of the least important non-zero DCT coefficient to avoid redundant computation [15]. For example, for the MB who does not have non-zero DCT coefficient, the reference MPEG-4 decoder skips its IDCT procedure, i.e. the processor cycles of IDCT procedure is zero. This situation is shown clearly in *Figure 3.6*.

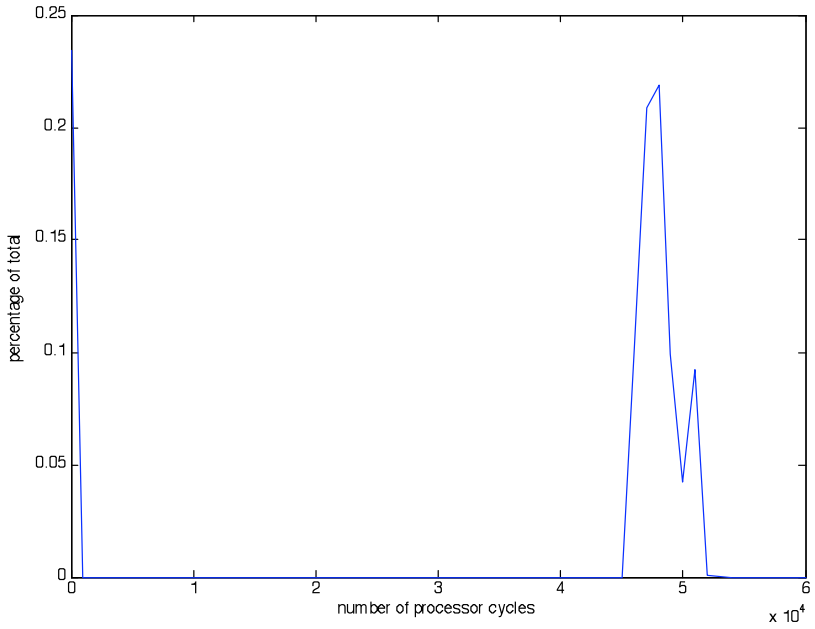

Figure 3.6 Processor cycles distribution of the IDCT task of reference MPEG-4 decoder

For future optimization, 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.

3.2.3 MC Task

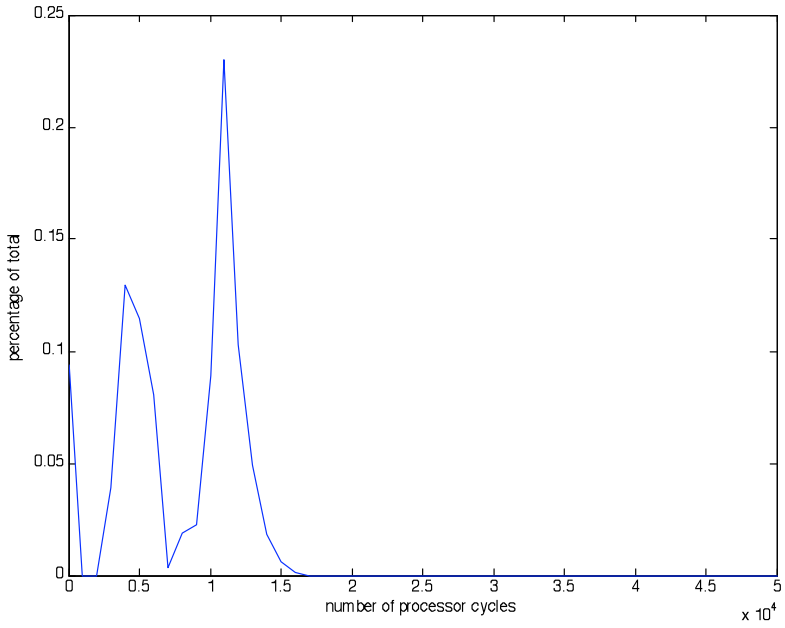

Figure 3.7 Processor cycles distribution of the MC task of the reference MPEG-2 decoder

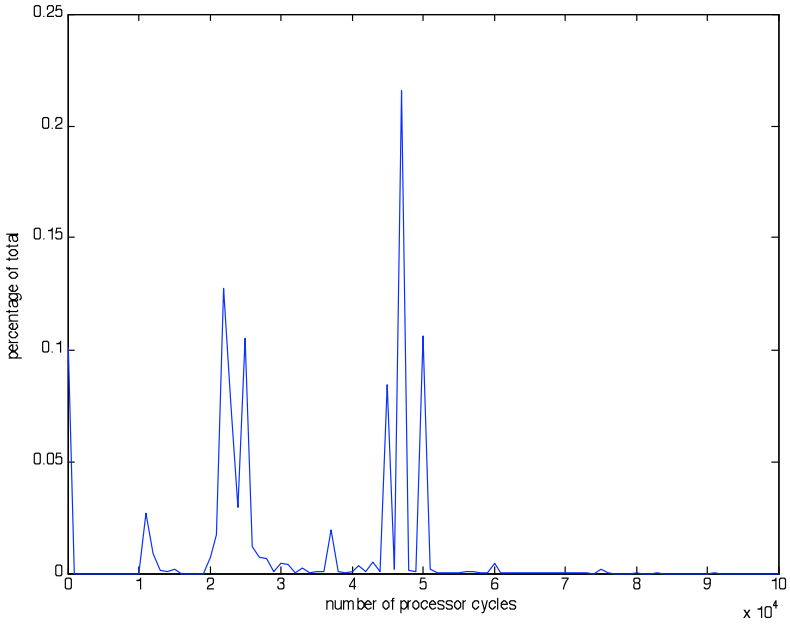

Figure 3.8 Processor cycles distribution of the MC task of the reference MPEG-4 decoder

For MC task, MBs can be classified into three categories: those that require no MC (I-type), those that require only forward MC (P-type) and those that require both forward and backward MC (B-type). Therefore, the MC task for P-type MBs incur about half the computational workload of B-type MBs while I-type MBs do not incur any computational workload. This thought is confirmed by *Figure 3.7* and *Figure 3.8*, which shows the processor cycle distribution of the MC task of the reference MPEG-2 decoder and the reference MPEG-4 decoder, respectively.

As expected, in both figures, the processor cycles are distributed into three distinct clusters. The first (around 0 processor cycles) corresponds to I-type MBs, the second (between 3000 - 7000 cycles for MPEG-2 decoder; between 10000 - 30000 cycles for MPEG-4 decoder) corresponds to P-type MBs, and finally the third (between 9000 - 17000 cycles for MPEG-2 decoder; between 43000 – 52000 cycles for MPEG-4 decoder) corresponds to B-type MBs. However, the processor cycle distribution within each cluster is very large. A modeling solely based on MB type will not be accurate. After analyzing the source code, we find that the MC task can also be divided into six sub tasks with each sub task as a MC operation for an 8×8 block. 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.

3.2.4 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 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.

3.3 Evaluation

In this section, we evaluate the workload model proposed before. In the experiments, we separate the video bitstream into two sets: 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 obtain the parameters of the workload model. Then, using the same parameters, we use the model to predict the workload of the video bitstreams in the testing set.

3.3.1 Experiment configuration

We run 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 (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) [10] and an optimized MPEG-4 in TCPMP project [1].

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

Table 3.1 12 CIF raw videos

On SimpleScalar, we measure the number of processor cycles as the decoding workload. On the IBM laptop and PDA, we measure execution time as the decoding workload. In the experiments, we have 12 CIF raw videos with different contents shown in *Table 3.1*.

Each of the video content is encoded in MPEG-2 and MPEG-4 format with four bit rates: 256 KBps, 512 KBps, 768 KBps and 1024 KBps. In total, we have 4 × 12 = 48 videos encoded for MPEG-2 and MPEG-4 format, respectively. The frame rate is set as 25 fps, and GOP size is set as 10. One or two B frames are inserted between the I- and P- frames. In the experiment, we divide the 48 encoded video bitstreams into 4 equal sets randomly: set A, B, C and D. We re-run each experiment 4 times. In each round, one set is picked as the testing set, and we build our workload model from the remaining 3 sets. Then the built model is used to predict the decoding workload of the 12 video bitstreams in testing set.

3.3.2 Results and Analysis

Figure 3.9 – Figure 3.14 show the experimental results of the workload prediction model. X-axis represents the prediction error rate, which is calculated by:

$$error\_rate = \frac{|actual\_workload - predicted\_workload|}{actual\_workload} \qquad (3.5)$$

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. *Figure 3.9* and *Figure 3.10* show the results on the IBM laptop in the first and the third run. *Figure 3.11* and *Figure 3.12* show the results on the SimpleSalar; and *Figure 3.13* and *Figure 3.14* 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.

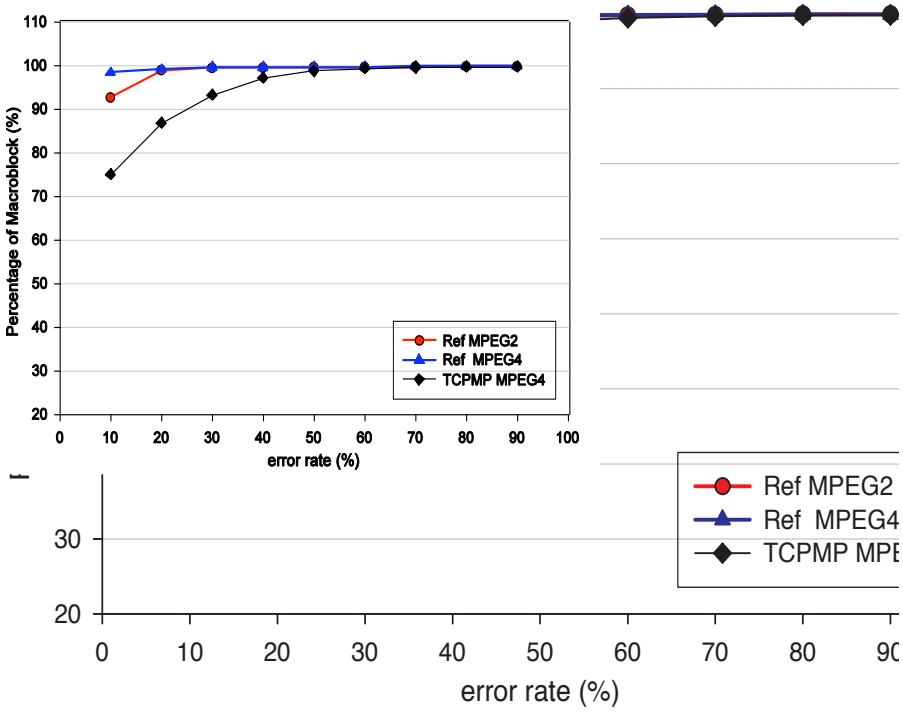

Figure 3.9 Cumulative prediction error rate of the decoding workload model, on Laptop (1st run)

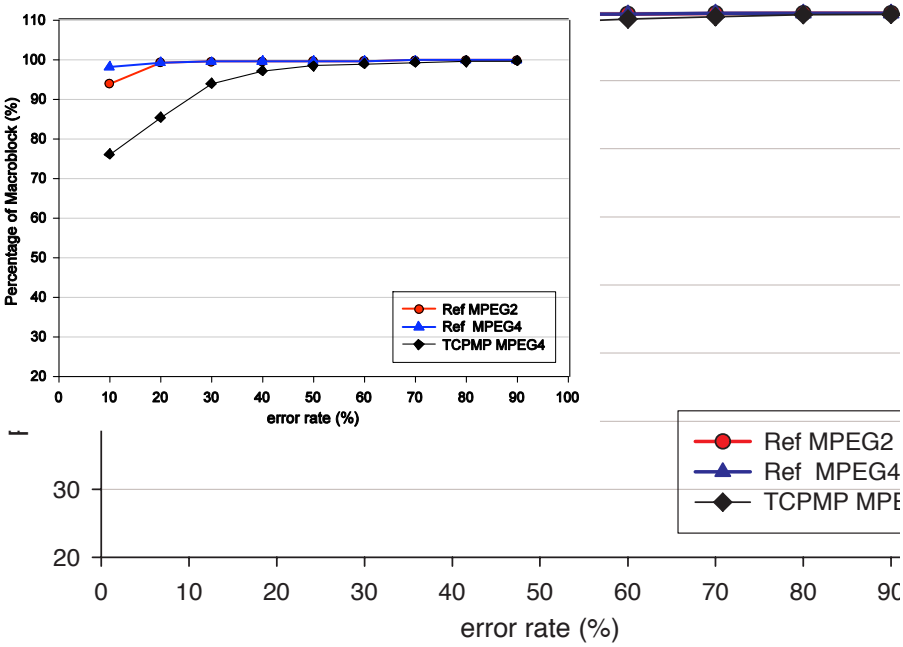

Figure 3.10 Cumulative prediction error rate of the decoding workload model, on Laptop (3rd run)

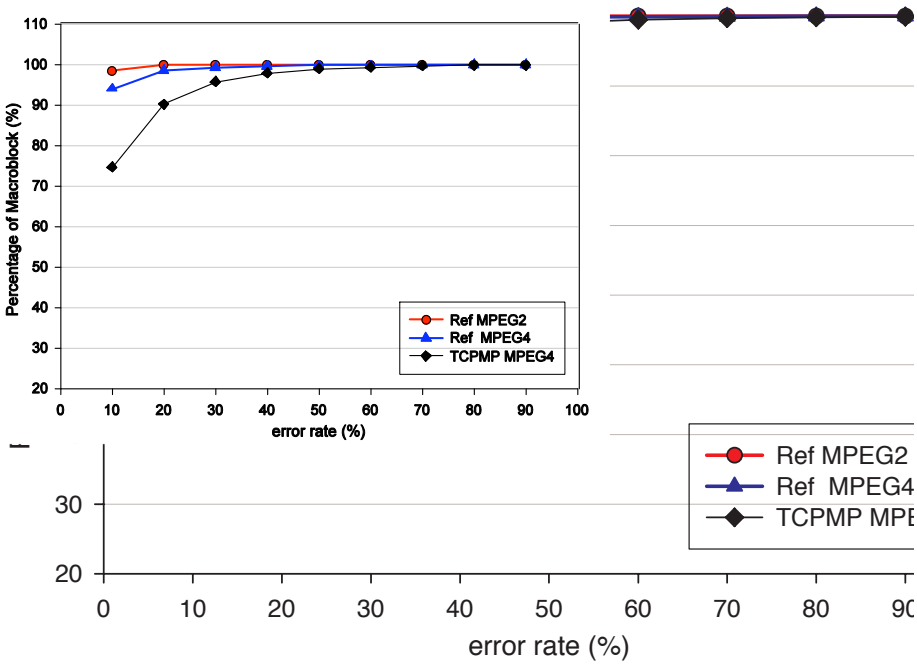

Figure 3.11 Cumulative prediction error rate of the decoding workload model, on SimpleScalar (1st run)

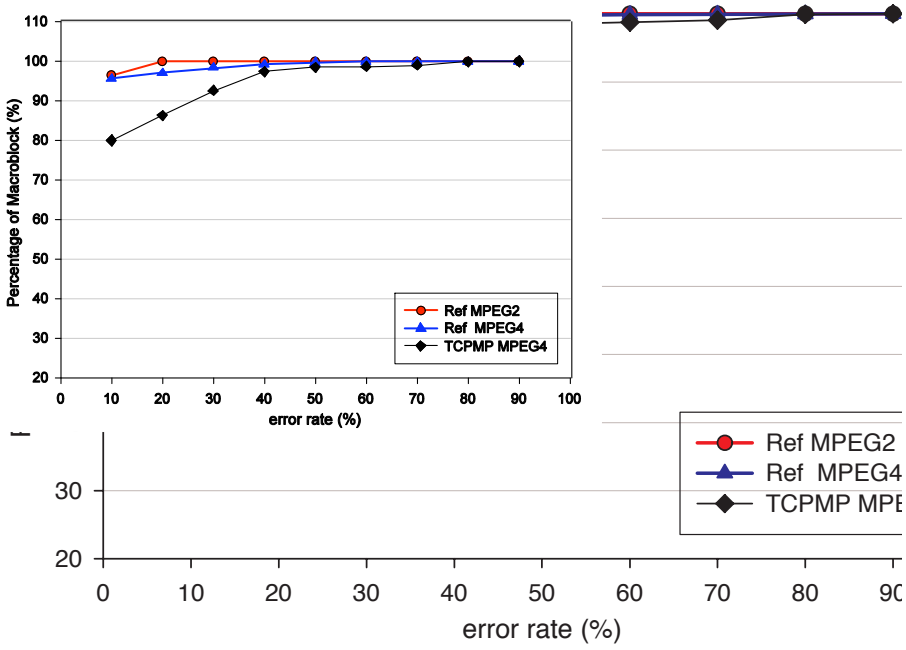

Figure 3.12 Cumulative prediction error rate of the decoding workload model, on SimpleScalar(3rd run)

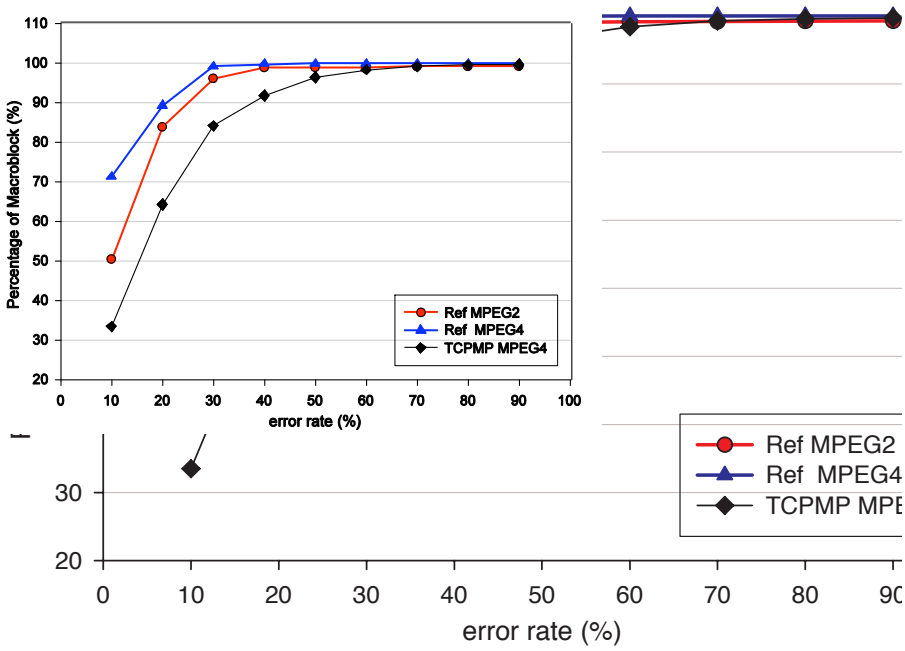

Figure 3.13 Cumulative prediction error rate of the decoding workload model, on PDA (1st run)

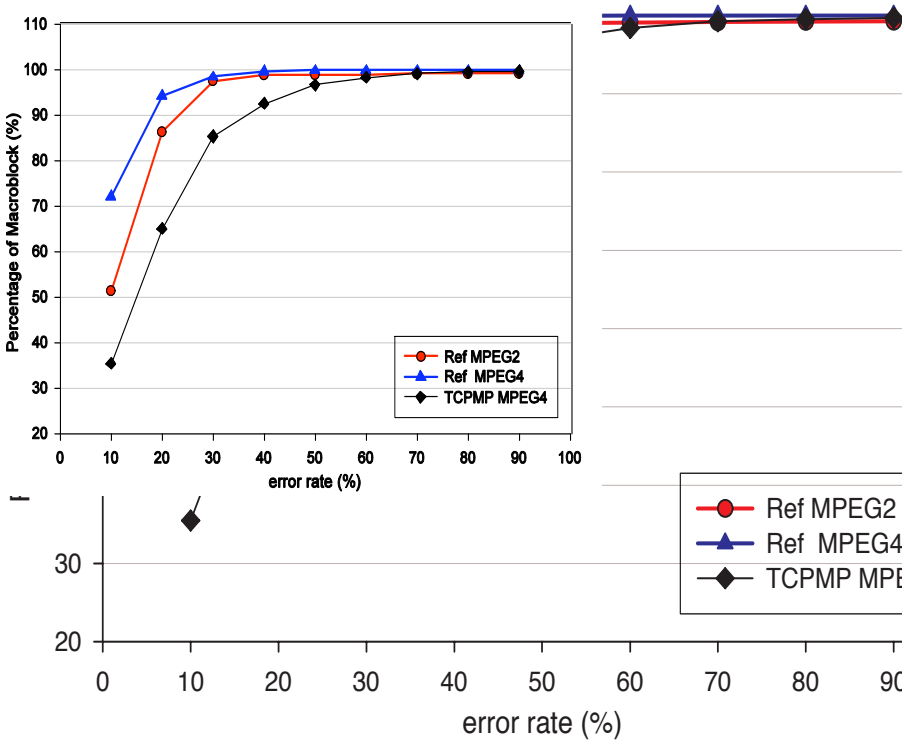

Figure 3.14 Cumulative prediction error rate of the decoding workload model, on PDA (3rd run)

The results show that on laptop and SimpleScalar, for both MPEG-2 and MPEG-4 reference decoder, more than 90% of MBs are predicted below an error rate of 10% and 98% of MBs are predicted below an error rate of 20%. But on the PDA, only 40% of MBs are predicted below an error rate of 10% and 90% of MBs are 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 show that the error mainly comes 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 varies significantly in cases of cache hits and cache misses. On PDA, the cache size is small, which causes many cache misses. This in turn makes the execution time less predictable. Since the PDA does not provide any mechanism for us to obtain the number of instructions, we can only use the execution time as the measurement for workload. This is why our model did not perform well on the PDA. On the laptop, the cache size is large, cache misses does not happen frequently. The execution time is not affected by the cache very much, so the model performed better. On SimpleScalar, we directly measure the number of instructions, which is not affected by the cache misses at all. That is why the prediction on SimpleScalar is 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 are 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.

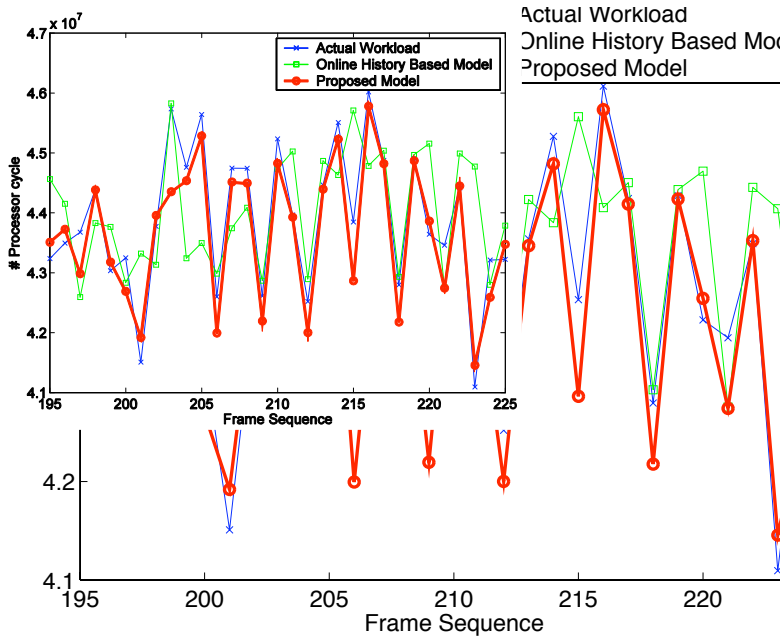

Figure 3.15 the comparison between our model and the history-based model

Figure 3.15 shows the comparison between our model and the history-based model proposed in [8]. 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 is 0.54 and the average error rate is larger than 20%. However, the correlation coefficient between our model and actual workload was 0.91 and the average error rate is less than 2%. This shows the advantage of our model.

3.4 Summary

In this chapter, we have presented a general decoding workload model. We verify the predictive power of this model by comparing it to the existing methods and actual workload measured on the device. We find 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 power embedded system design and many other relevant tasks.

Chapter 4

Workload-Scalable Transcoder

4.1 Introduction

In this chapter, we present the workload-scalable transcoder application. We assume such a scenario: 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 4.1* shows the architecture of our proposed scheme, where a transcoding proxy employed in 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.

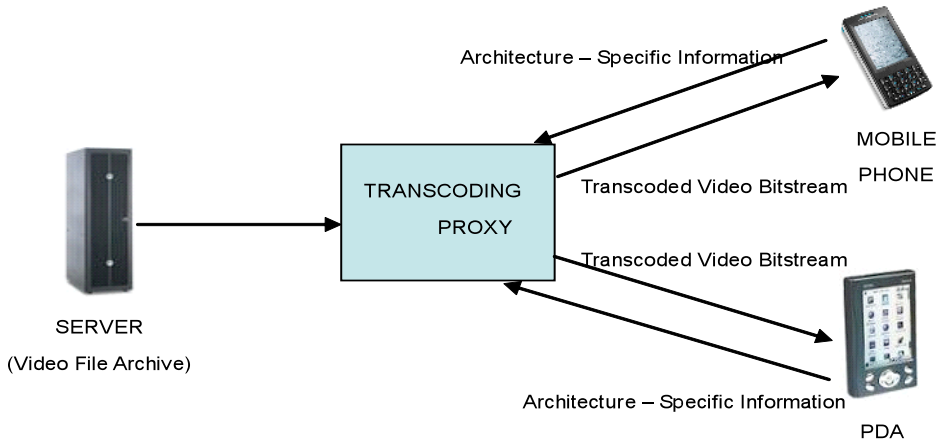

Figure 4.1 System architecture for the transcoding scheme

As mentioned in Chapter 3, 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-encoders it. This is very time consuming [13]. Our transcoding scheme is designed to operate in the compression 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 and motion vectors during transcoding; therefore this known information can be used to control the target workload.

The challenges are: 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. The first problem can be addressed by using the decoding workload model proposed in Chapter 3. Using the model, we can predict the decoding workload based on information extracted from the video bitstream. To solve the second problem, we propose a measure, called mean compression domain error (MCDE), to estimate the video quality for all the candidates quickly.

The proposed transcoding scheme，as shown in *Figure 4.2* is done in the following three steps:

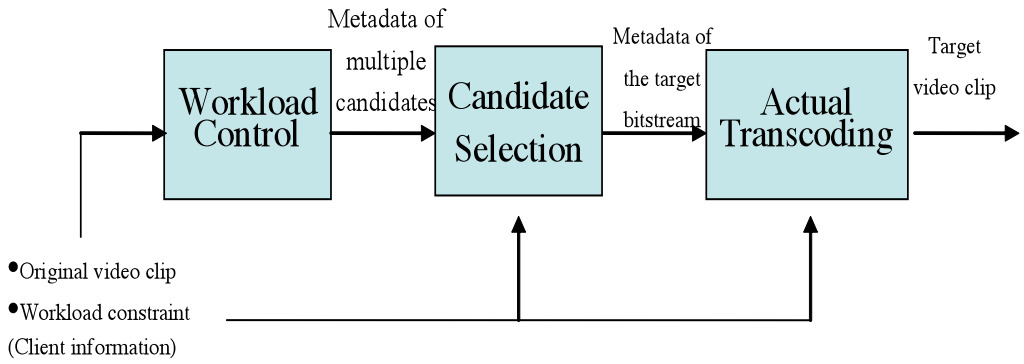

Figure 4.2 Transcoding Scheme

Workload Control: Given the constraint, the decoding workload is reduced by decreasing the frame rate and dropping the Huffman codes. This step may generate more than one candidates having the workload below the workload constraint. It is noted that we do not

do any actual transcoding in this step. The output of this step is the candidates' metadata, which is the file indicating which frames and Huffman codes should be dropped. The details of this step will be presented in Section 4.2.

Candidate Selection: In this step, we use MCDE to estimate the video quality of all the candidates. We select the candidate with minimal distortion from the original video bistream as the final result, whose metadata is then feed into step 3. The details of MCDE will be presented in Section 4.3

Actual Transcoding: the transcoder reads the metadata and performs the actual transcoding of the original video bitstream to the target video bitstream. This step is quite straightforward and we will not explain it more.

The contributions of this chapter are two-fold:
We propose a measure called mean compression domain error (MCDE), which can estimate the quality of the transcoded video with very low computational complexity. Based on the decoding workload model proposed in Chapter 3 and MCDE, we propose a three-step compression domain transcoding scheme, which can accurately control the decoding workload of the target video bitstream and keep the distortion between the target and original video bitstream minimal.

The rest of this chapter is organized as follows. In Section 4.2, we present the transcoding scheme. We introduce MCDE in Section 4.3. In the evaluation is presented in Section 4.4. We summarize the work in Section 4.5.

4.2 Workload Control Scheme

In the step of *Workload Control*, we reduce the decoding workload of the video bitstream by decreasing the frame rate and dropping the 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 fps), the number of the 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 Chapter 3, decisions are made as to which Huffman codes should be discarded for the remaining frames. The details are shown in *Algorithm 4.1*.

Algorithm 4.1: Workload Control

```
Input: Target Workload (InputTarWL)
Output: Metadata for candidate video bitstreams
GetOrigInfo(); /*to get necessary information from the original video bitstream */
foreach FrameRate fr do
        DropFrame(fr);
        if (MinReqWL(fr) >= InputTarWL) then
                Iterate for the next possible frame rate.
        end
        TotalTarWL = InputTarWL
        foreach remaining frame f_curr do
                AllocFrameWL (f_curr)
                DiscardHuffman(f_curr)
                Update(TotalTarWL)
        end
end
```

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.

MinReqWL(fr):

The minimum decoding workload request for the frame rate *fr*. It is calculated as the sum of the MC task's workload of the rest frames plus the safe margin.

AllocFrameWL(f_{curr}):

For the current frame rate f_r, we denote the N frames which are kept after *DropFrame(fr)* as $f_0, f_1, \ldots, f_{N-1}$. Let f_{curr} be the current frame. The decoding tasks of each MB are divided

into two parts: *Huff Comp* includes the tasks whose workload depends on the number of Huffman codes; *NonHuff 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 *GetOrigInfo()* of *Algorithm 4.1*. We denote *TotalTarHuffWL* the total target workload of the Huff Comp. It is calculated as:

$$TotalTarHuffWL = TotalTarWL \sum_{i=f_{curr}}^{f_{N-1}} OriNonHuffWL[i] \qquad (4.1)$$

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

$$TarHuffWL[f_{curr}] = OriHuffWL[f_{curr}] * \frac{TotalTarHuffWL}{\sum_{i=f_{curr}}^{f_{N-1}} OriNonHuffWL[i]} \qquad (4.2)$$

DiscardHuffman(f_{curr}):

The details of this function is shown in *Algorithm 4.2*. In *Algorithm 4.2*, function *Discard(DCT_Pos)*, "the Huffman codes after *DCT_Pos*" are the Huffman codes whose position is 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.

Algorithm 4.2 DiscardHuffman

Input: *TarHuffWL[f_{curr}]*
Output: Metadata, assigned workload
if (*TarHuffWL[f_{curr}]>= 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 <= TarHuffWL[f_{curr}]*) **then**
        return *Huff_WL*
    end
end

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:

$$TotalTarWL = TotalTarWL - OriNonHuffWL[f_{curr}] - TarHuffWL[f_{curr}] \quad (4.3)$$

4.3 Mean Compression Domain Error

MCDE is proposed for those compression domain transcoder application. Generally speaking, there are two popular methods for a compression domain transcoder to reduce the video bit rate and decoding workload, namely reducing the bit per frame (bpf) and frame per second (fps). Reducing bpf increases spatial distortion while reducing fps increases temporal distortion. For a given constraint, there could be multiple candidates

with different combinations of spatial quality (bpf) and temporal quality (fps). Thus, an objective video quality measure which can predict the overall video quality considering both spatial and temporal distortions becomes a critical component.

Conventional measures such as PSNR and MSE operate in the pixel-domain, which require full decoding of both original and candidate video bitstreams and are computationally too expensive for real-time transcoding applications.

Fortunately, the work in [7, 14] provides a solution. They replace the dropped frames by copying the previous frames in the display order. The rationale is that a player can maintain the current frame on the screen before displaying the next frame. However, they still use PNSR or MSE to estimate the distortion between individual frames. This demanded that the system actually generate all the transcoded bitstream and decode them to the spatial domain, which is too expensive especially when the transcoder itself works in the compression domain. In the proposed MCDE, we use a similar approach. However, the distortion between two frames is calculated in the compression domain. Then the MCDE is calculated as the average distortion between the original and transcoded frames.

It is noted that the distortion of the remaining frames (after frame dropping) can be regarded as the spatial distortion and the distortion of the replaced frames can be regarded as the temporal distortion. To simplify the problem, we analyze the two types of distortion separately and then combine them to produce the overall distortion. 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.

4.3.1 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. However, for P- and B-frames, error propagation has to be considered as well. It is because the frames that P- and B-frame depend on could also be distorted. In our measure, the spatial distortions caused by dropping Huffman codes for different types of frames are estimated by the following equations:

For I-frame

$$D_S(I, I') = H(I) - H(I') \qquad (4.4)$$

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

For P-frame

$$D_S(P, P') = \alpha D_S(F, F') + (H(P) - H(P')) \qquad (4.5)$$

where P and P' are the original and transcoded frames; F and F' are the frames P and P' depend on, respectively; α is a parameter presenting the effect of error propagation.

For B-frame

$$D_S(B,B') = \alpha(D_S(F_1,F_1') + D_S(F_2,F_2'))/2 + (H(B) - H(B')) \qquad (4.6)$$

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; α is the same parameter as in Equation 4.5.

4.3.2 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 un-dropped frame. We calculate the distortion for every individual frame and sum the result up as the distortion of 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'. By our assumption, since 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:

$$D_T(P_1, P_2) = H(P_2) \qquad (4.7)$$

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

$$D_T(P_0, P) = \alpha D_T(P_0, P_1) + D_T(P_1, P) = \alpha D_T(P_0, P_1) + H(P) \qquad (4.8)$$

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. α (the same parameter in Equation 4.5) is the parameter representing the effect for error 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:

$$D_T(B, P_0) = \alpha(D_T(P_0, P_1) + D_T(P_0, P_2))/2 + H(B) \qquad (4.9)$$

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. α (the same parameter in *Eq 4.2*) is the parameter representing the attenuation effect for error 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:

$$D_T(B, B_0) = H(B_0) + H(B) \qquad (4.10)$$

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:

$$D_T(I, I_0) = H(I_0) + H(I) \qquad (4.11)$$

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

4.3.3 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:

$$D(F, F_0^{'}) = wD_S(F_0, F_0^{'}) + (1-w)D_T(F, F_0) \qquad (4.12)$$

where w is the weight between spatial distortion and temporal distortion. The average of the distortion of all the original and their transcoded frames is calculated as the final MCDE.

MCDE can be summarized as 1) we use difference of the number of the Huffman codes to estimate the spatial error. 2) We use the number of Huffman codes of P/B frame to estimate the temporal error. 3) We consider the error propagation and 4) we linearly combine the temporal and spatial distortion to estimate the distortion between the original frame and the dropped frame.

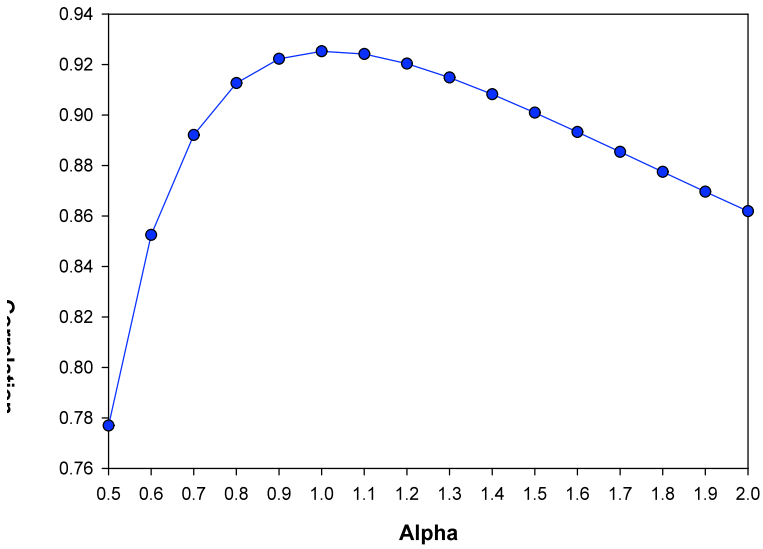

Figure 4.3 The correlation between MCDE and subjective result with different α values

There are two parameters in MCDE, w and α. It is difficult to select an optimal value for w, because the optimal value can be different for different video content. For example,

when the movement of the video is low, the spatial distortion is more important, thus w should be small, and vice versa. In our scheme, considering the balance for all the cases, w is set to 0.5.

To choose the value for α in Equation 4.5, 4.6, 4.8 and 4.9, we conduct the experiments varying α from 0.1 to 2.0 (with w is fixed as 0.5). For each value of α, we compare the MCDE and the subjective results, which will detailed introduced in the next section. The comparison is shown in *Figure 4.3*. And we can see that when α is set to 1.0, the correlation between MCDE and subjective result is the largest.

It is also noted that when estimating the spatial distortion, we assume that all Huffman codes, i.e. all DCT coefficients, are equally important. The reason is that MCDE is only designed to select the best candidate rather than as a generic video quality measure. Our experiments have shown that the proposed MCDE already performs well (without considering the different significance of different Huffman codes.

4.4 Evaluation

4.4.1 Mean Compression Domain Error Evaluation

In this subsection, we evaluate the MCDE's performance.To evaluate its accuracy, we compare the MCDE with MSE and stimulus continuous quality scale (DSCQS) [35] in Section 4.4.1.1; to evaluate its speed, we compare the computational complexity between MCDE and MSE in Section 4.4.1.2.

4.4.1.1 Comparison among MCDE, MSE and DSCQS

In our experiments, we have three original CIF-size MPEG-4 video clips, which are shown in *Table 4.1*:

Table 4.1 Video sequence used to compare MCDE, MSE and DSCQS

Name	Bit rate	Descriptions
Hall_768	768 KBps	Still background and two objects with moderate movements
Highway_1024	1024 KBps	Moving background
Walk_512	512 KBps	Both background and two foreground objects are with very fast movements

Each of them is transcoded using different configurations. First, we fix the target frame rate as 8fps and 15fps and vary the number of Huffman coefficients as one of 10%, 20%, 40%, 60%, 80% and 100% of that of the original video clip. Then we fix the number of Huffman coefficient as 30% and 50% of the original video clip and vary the target frame rate as one of 5fps, 8fps, 12fps, 15fps, 20fps and 25fps. Thus, totally we have $3\times2\times6+3\times2\times6=72$ transcoded video clips for testing.

For each transcoded video clip, we calculate its MCDE and MSE. We also evaluate them using subjective testing. The 72 video clips are divided into three groups, and the video clips in each group have the same content. Thirty normal-eyesight viewers are invited to

our test. Each of them evaluates one group of video clips. We select double stimulus continuous quality scale (DSCQS) as our subjective video quality methodology [35]. In DSCQS, the viewers are shown pairs of video clips (the original clip and the transcoded clip) in a randomized order. Each pair is displayed twice. After the second display, viewers are asked to rate the quality of each clip in the pair. The difference between these two scores is then used to quantify changes in quality. [35]

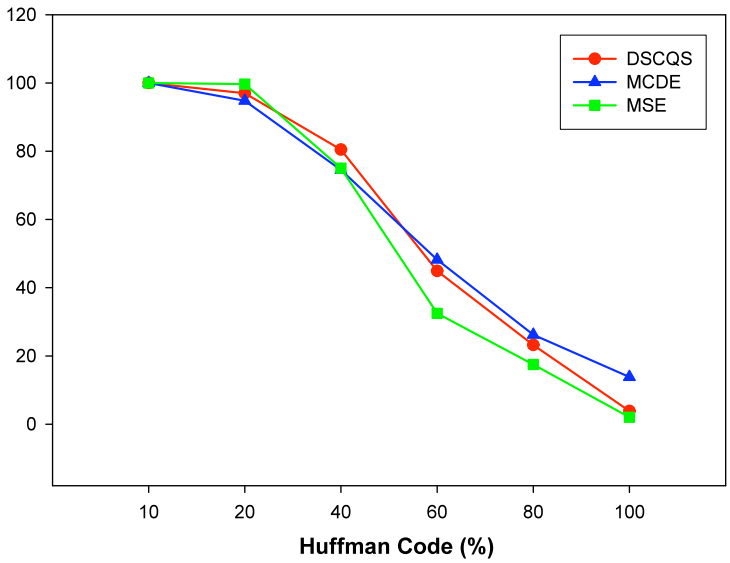

Figure 4.4 comparison among MCDE, MSE and DSCQS for Hall_768 with 15fps

Figure 4.4 shows the comparison among MCDE, MSE and DSCQS for different percentages of the number of Huffman codes with the same target frame rate (15fps). The y-axis represents the quality distortion after normalization. The x-axis represents the percentages Huffman codes of the original video clips. It is observed that all MCDE, MSE and DSCQS decrease as the number of Huffman codes increases. The three curves follow the same trend.

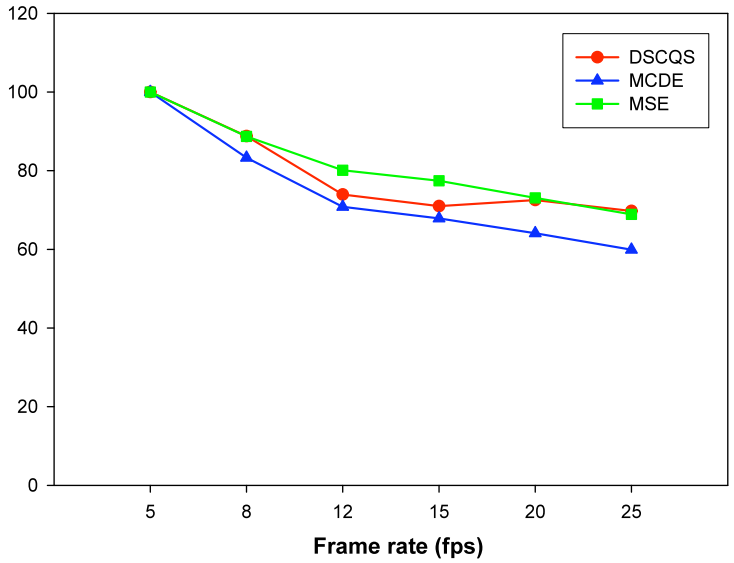

Figure 4.5 comparison among MCDE, MSE and DSCQS for Highway_1024 with 50% Huffman codes

Figure 4.5 shows the comparison among MCDE, MSE and DSCQS for different frame rates with the same number of Huffman codes (50% of original). The y-axis represents the quality distortion after normalization. The x-axis represents the frame rate. It is observed that the curves of MCDE and MSE follow the same trends. Both of them decrease as the frame rate increases. However DSCQS increases as the frame rate increases from 15fps to 20fps. The similar results are also found in other subjective testing groups. It might be because that it is hard for people for to distinguish the temporal difference when the frame rate is larger than 15fps. This exactly matches the results obtained in [3].

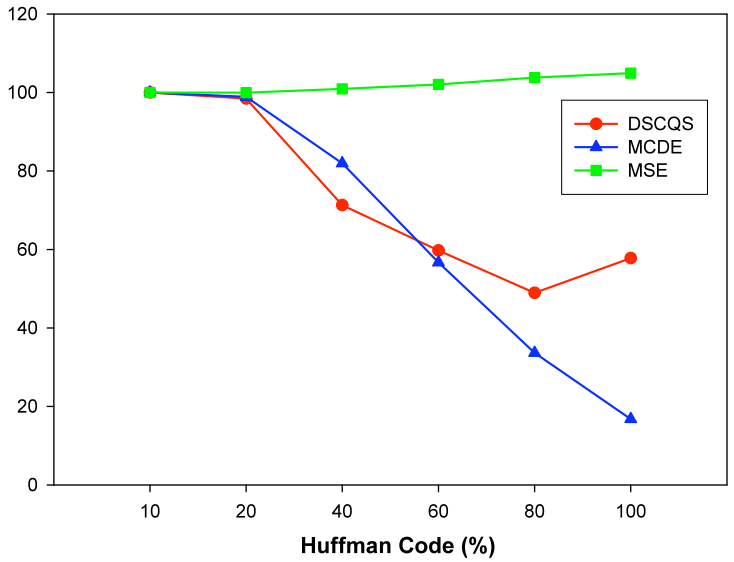

Figure 4.6 Comparison among MCDE, MSE and DSCQS for Walk_512 with 8fps

Figure 4.6 shows the comparison among MCDE, MSE and DSCQS for different number of Huffman codes with the same target frame rate (8fps). In this figure, only MCDE decreases as the number of Huffman codes increases. DSCQS almost has the same trend with MCDE except it increases as the number of Huffman codes increases from 80% to 100%. It is probably because for this video clip, the spatial distortion between 80% and 100% are very close. It is hard for people to distinguish them. MSE increases as the Huffman codes increases. It may be because 'Walk_512' has very fast motion, when the frame rate is low, MSE cannot measure the distortion correctly. In such a case, MCDE is even more accurate than MSE.

4.4.1.2 Computational Cost between MCDE and MSE

We also measure the computational complexity for both MSE and MCDE. Given the information of how to drop frames and Huffman codes, to calculate MSE we need to 1) actually transcode the video clip, 2) decode both original and transcoded video clips, and

3) calculate MSE. On average, that costs about 5 seconds for a 10-second video clip, on a Pentium 4, 3GHz, 1G RAM PC. On the other hand, the calculation of MCDE only takes around 0.5 seconds on the same PC. It is noted that we implement MSE using C++ while MCDE using Python. Although Python is much slower than C++, the calculation of MCDE is still 10 times faster than MSE.

4.4.2 Transcoding Scheme Evaluation

In this subsection, we evaluate the transcoding scheme. We transcode the existing video bitstreams under the different decoding workload constraints. The transcoding scheme is evaluated with the following two aspects: 1) whether *Workload Control* scheme can accurately control the target decoding workload, which should be just below the given constraint; 2) whether MCDE can actual select the best candidate. This can be proven by whether the estimation result matches that of the traditional PSNR result.

To evaluate the first aspect, we measure the actual decoding workload of the target video bitstream and compared it to the original workload constraint. To evaluate the second aspect, we decode all the candidates generated by *Workload Control* `model` and calculate the actual PSNR between the candidates with the original video bitstream. We then check if the result we select using MCDE had a highest PSNR (the higher the average PSNR is, the less distorted the transcoded video bitstream is compared to the original video bitstream).

4.4.3 Experiment configuration

The same 12 raw video described in Table 3.1 (and each encoded by 4 bit rates) are used in this evaluation. We run experiments for 8 different workload constraints corresponding to the processor frequencies 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 are set to be 5, 8, 10, 15, 18, 20, 22 or 25 fps. The actual workload is measured on SimpleScalar.

4.4.4 Workload Control Evaluation

Figure 4.7 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 4.7* shows how accurately our transcoding scheme could control the workload of the transcoded video bitstreams. The curve labeled *Workload Constraint* represents the constraints. The curves labeled *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 256 KBps, 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 500 MHz. This was because that processor frequency of 500 MHz is more than enough to decode the original video bitstreams.

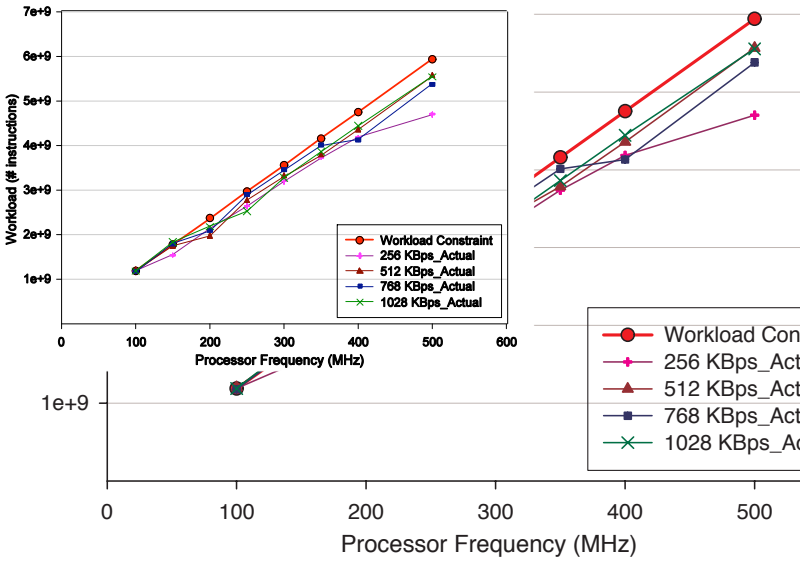

Figure 4.7 The comparison for the actual decoding workload and workload constraint

4.4.5 Candidate Selection Evaluation

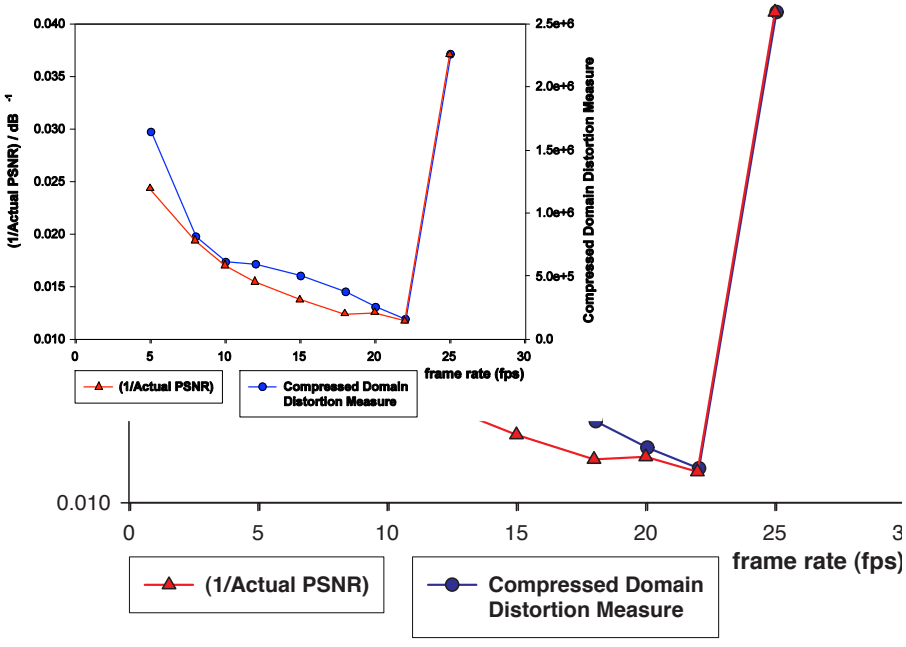

Figure 4.8 Comparison between the MCDE and 1/Actual PSNR

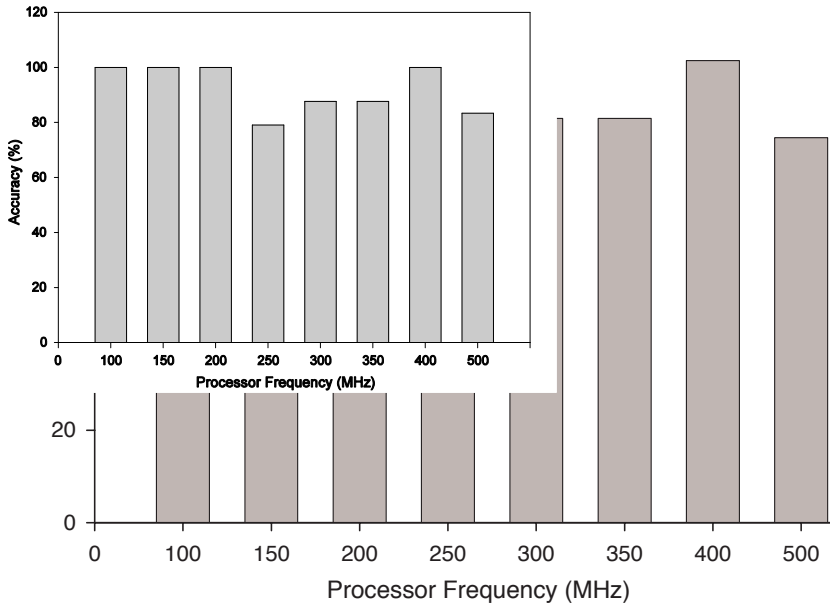

Figure 4.9 Accuracy of the candidate selection

After the *Workload Control* step, we had the metadata of all possible candidates. In this experiment, we perform the transcoding of all these video bitstreams and calculate the actual PSNR from the original video bitstream. Using MCDE, we estimate the distortion of these transcoded video bitstreams from the original, and then we compare the estimated distortion values with the (1/Actual PSNR) values. *Figure 4.8* shows a comparison between the MCDE 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 MCDE and actual PSNR. *Figure 4.8* shows that MCDE 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 selects the candidate with lowest (1/Actual PSNR) value, the selection is correct; otherwise, the selection was wrong. *Figure 4.9* shows the accuracy of the candidate selection for different processor

frequencies. On average, in more than 92% out of 192 experiments, our estimation algorithm selects the best quality video bitstreams, and in the rest 8%, the second best quality video bitstreams are selected.

4.5 Summary

In this chapter, we design a workload-scalable transcoder to provide an optimal match between the transcoded bitstream and a mobile device's processing power. The scheme takes advantage of MCDE and the decoding workload model proposed in Chapter 3: MCDE is used to find the best candidate; the decoding workload model is used to control the workload of the transcoded bitstream. Both components have been evaluated with experiments and were shown to be effective. The main advantage of our compression 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.

MCDE is a new objective video quality measure for the transcoding applications. Our experiments show that MCDE can be used to accurately predict the subjective quality of the transcoded video with negligible computational complexity in comparison with the conventional MSE. It is noted that although MCDE is proposed to solve the problem of our transcoder, it can also be applied to other compression domain transcoding applications.

Chapter 5

Workload Scalable Encoder

5.1 Introduction

In this chapter, we present workload-scalable encoder. The workload-scalable encoder is analog to the workload-scalable transcoder presented in the last chapter: both of them generate the target bitstream according the decoding workload constraints of the mobile device; both of them need to consider the tradeoff between frame rate and individual frame quality. The difference is the transcoder constructs the bitstream from the existing bitstream while the encoder constructs the bitstream from the raw video data. This difference makes the mechanism of the encoder completely different from that of the transcoder:

- In the transcoder, the decoding workloads are allocated to the frames/MBs according to their original decoding workload. In the encoder, we do not have the 'original decoding workload'.

- In the transcoder, the distortion of the different frame rate candidates are estimated by the difference of the original and target bitstream's Huffman codes numbers. In the encoder, original video is raw data, which does not have the Huffman codes.

Therefore, for the workload-scalable encoder, we have to re-consider both problems: decoding workload control and frame rate selection. For the decoding workload control, we can still take advantage of the decoding workload model to control decoding workload by adjusting the parameters of the bitstream during the encoding. However, how to allocate decoding workload to different frames/MBs is a problem. As mentioned in Chapter 1, even with the same quality, different frames/MBs require different decoding workloads. The encoder should smartly allocate a decoding workload so that different frames/MBs have the same quality. For the frame rate selection, the encoder requires an objective measure to estimate the video distortion for different frame rate candidates. A conventional approach is to encode all the possible candidates, then use the approach in [7, 28] to calculate the average MSE/PSNR for them. However, this approach is very expensive. In this chapter, we propose a new frame rate selection scheme, which can estimate the video distortion before the actual encoding. Our experiment shows that the scheme can get a similar or even better result than the conventional approach mentioned above but with much lower computational complexity.

The architecture of the encoder is shown in *Figure 5.1*.

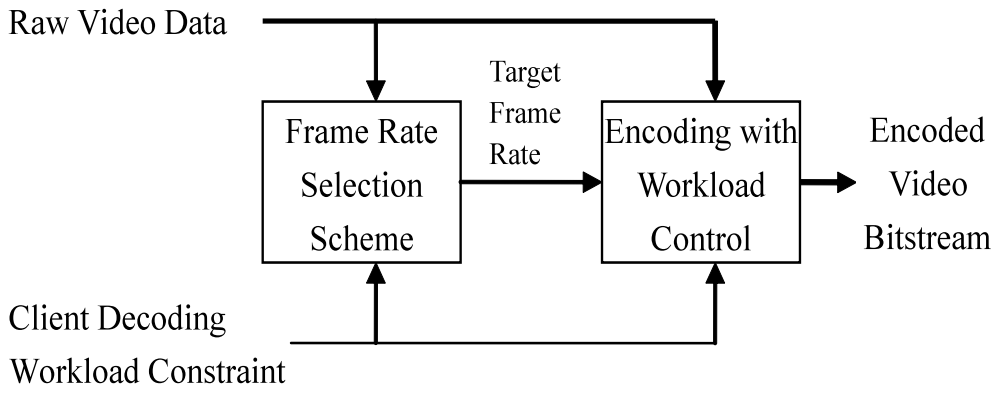

Figure 5.1 The encoder architecture

The encoding procedure includes two phases. In the first phase, the raw video data and client decoding workload constraint together with all possible frame rates are fed into the *Frame Rate Selection Scheme* which selects the most suitable frame rate for the actual encoding. In the second phase, the encoder uses the selected frame rate and client decoding workload constraint to compress the raw video into the target video bitstream using the *Workload Control Scheme*. We also assume that the GOP format is fixed.

The contributions of this chapter are two-fold:

We propose Frame Rate Selection scheme which can select the most suitable frame rate before the actual encoding with very low computational complexity. It is noted that although this scheme is proposed for the workload-scalable encoder, its concept can also be used to other encoder applications.

We propose Workload Control scheme which can accurate control the decoding workload of the encoded bitstream. It employs smart strategies to allocate the workload

so that under the same decoding workload, it can generate a bitstream with better video quality (compare to the encoder does not employ the strategies).

The rest of the chapter is organized as the follows: In Section 5.2, we introduce the frame rate selection scheme. The workload control scheme is presented in Section 5.3. We evaluate the schemes in Section 5.4 and summarize this chapter in Section 5.5.

5.2 Frame Rate Selection Scheme

This section presents our fast frame rate selection scheme: it enumerates all the frame rate candidates; for each candidate, the distortion of the target video bitstream is estimated. The candidate with the smallest distortion is selected. The problem is how to perform a fast distortion estimation of all target video bitstreams with all possible frame rates before actual encoding.

Before going to the detail of the algorithm, we introduce some notations first. Assume we have a raw video sequence containing N frames: $P(0), P(1), P(2)…P(N-1)$ (see *Figure 2*). For each frame rate candidate f, we evenly select $M=N*f/f_{max}$ frames from the original sequence for actual encoding, where f_{max} is the maximum frame rate. In our implementation, f_{max} is set as 25fps. We denote $P'(0),P'(1),P'(2),…,P'(M-1)$ are the frames decoded at the client end. In *Figure 2*, f is equal to 12 fps. Replacing a dropped frame by its previous frame, we get the frame sequence $P'(0,0),P'(0,1)..P'(0,f_{max}/f-1),P'(1,0),P'(1,1)…P'(1,fmax/f-1)…P'(M-1,0),P'(M-1,1)…P'(M-1, f_{max}/f-1)$, where $P'(i,j)$

is exactly the same as *P'(i,0)*. And *P'(i,j)* is corresponding to the frame $P(i*f_{max}/f+j)$ in the original video sequence.

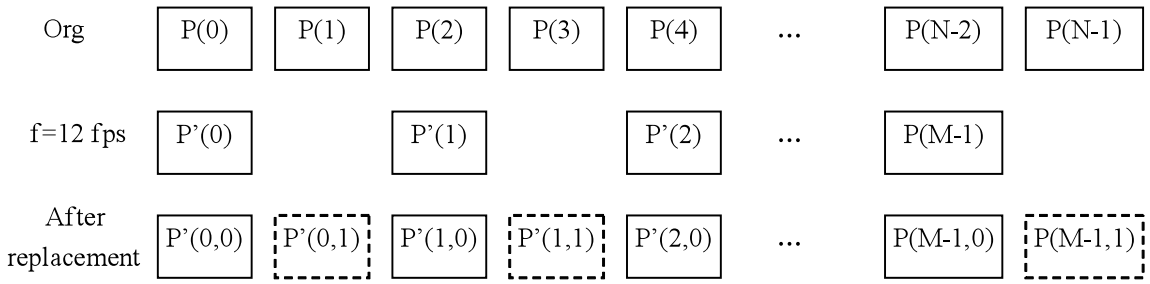

Figure 5.2 An example case for frame rate selection scheme

The distortion of the video sequence with frame rate *f* is calculated by the distortion of the video sequence after frame replacement, which is then calculated by the average distortion of the corresponding frames. Here are the basic ideas to calculate the distortion between two corresponding frames:

For the frames *P'(i,0)*

According to the frame rate and GOP structure, we know the frame type of *P'(i,0)*. Using a simple version of workload control scheme in Section 2, we can also estimate the number of Huffman codes in this frame.

if P'(i,0)is an I-frame

The variance of the frame describes its image complexity. The distortion between *P'(i,0)* and *P(i* f_{max}/f)* is estimated as the image complexity lost when being encoded into the target bitstream due to workload constraint, i.e., a MB will be coded using just a part of the total 64 Huffman codes:

$$D(i,0) = Var(i * f_{\max} / f) * (1 - \frac{w_{huff}(N)}{w_{huff}(64)}) \qquad (5.1)$$

where *N* is the number of Huffman codes, $W_{huff}(N)$ is the weight of the first N Huffman codes, *Var(i)* is the variance of the original frame, which can be calculated before the actual encoding.

if P'(i,0) is an P-frame

A P-frame is dependent on its reference frame. Assuming the reference frame is *P'(k)*, its distortions have two parts: the distortion propagated from its reference frame and the residual error lost due to the workload constraint:

$$D(i,0) = W_{prop} * D(k,0) + \mathrm{Res}'(k,i) * (1 - \frac{w_{huff}(N)}{w_{huff}(64)}) \qquad (5.2)$$

where W_{prop} is the weight representing the error propagation effect, *D(k, 0)* is the distortion of *P'(k)* which can be calculated by Eq. 5.1, if *P'(k)* is an I-frame; or by 5.2, if *P'(k)* is a P-frame. *Res'(k,i)* is the residual error between *P'(k)* and *P'(i)*. The residual error is calculated in the motion compensation procedure. Since running the motion

compensation for all frame rate candidates is computationally expensive, we estimate *Res'(k,i)* by

$$\mathrm{Re}s'(k,i) = \mathrm{Re}s(p,q) = \mathrm{Re}s(p,p+1) + W_{res} * (\mathrm{Re}s(p+1,q)) \qquad (5.3)$$

where $p=k*f_{max}/f$, $q=i*f_{max}/f$, W_{res} is a parameter. Thus, we can estimate the residual error between any two frames by a linear combination of the residual error between two adjacent original frames, which needs to be calculated only once before the actual encoding.

if P'(i,0) is an B-frame

It depends on two frames *P'(k)* and *P'(t)*. Similar to the P-frame, its distortion can be calculated as:

$$D(i,0) = W_{prop} * \frac{D(k,0) + D(t,0)}{2} + \frac{\mathrm{Re}s'(k,i) + \mathrm{Re}s'(i,t)}{2} * (1 - \frac{w_{huff}(N)}{w_{huff}(64)}) \qquad (5.4)$$

For the frames P'(i,j), where j>0

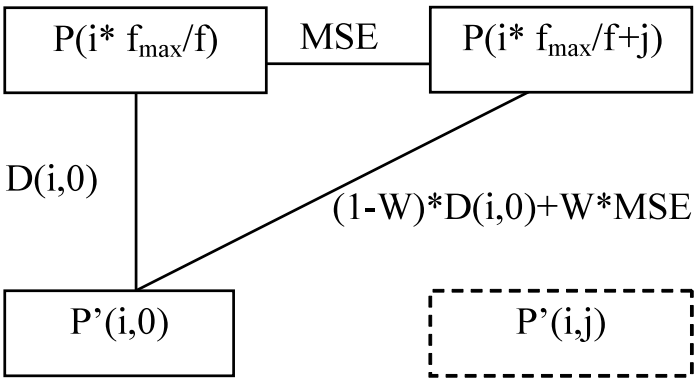

Figure 5.3 the distortion calculation for P'(i,j)

The distortion between *P'(i,j)* and its corresponding frame $P(i*f_{max}/f+j)$ equals to the distortion between *P'(i,0)* and $P(i*f_{max}/f+j)$, since *P'(i,j) is a direct copy of P'(i,0)*. Using the frame $P(i*f_{max}/f)$ as a bridge (see *Figure 3*), the distortion can be estimated as a weighted sum of the spatial distortion between *P'(i,0)* and $P(i*f_{max}/f)$, and the temporal distortion between $P(i*f_{max}/f+j)$ and $P(i*f_{max}/f)$.

$$D(i,j) = (1-W_{temp})*D(i,0)+W_{temp}*MSE(i*f_{max}/f, i*f_{max}/f+j) \qquad (5.5)$$

where W_{temp} is the weight of the temporal distortion and $MSE(i*fmax/f, i*fmax/f+j)$ is the MSE between $P'(i*fmax/f)$ and $P(i*fmax/f+j)$, which is used to represent the temporal distortion caused by the frame replacement in the display sequence. Again, we do not want to calculate MSE for all the possible candidates. Let $p=i*f_{max}/f$ and $q=i*f_{max}/f+j$, we estimate *MSE(p,q)* by

$$MSE(p,q) = MSE(p,p+1) + W_{mse} * MSE(p+1,q) \qquad (5.6)$$

Thus, we estimate the MSE between any two frames by a linear combination of the MSE between two adjacent original frames, which needs to be calculated only once before the actual encoding.

The detail of the distortion estimation for each frame rate candidate is shown as *Algorithm 5.1*.

Algorithm 5.1. Frame Rate Selection Scheme

1) Select the frames, $P'(i,0), i=1\ldots M,$ from original sequence based on the frame rate.
2) For all the selected frames:
3) Allocate the workload to the current frame according to the frame type.
4) Estimate the sum of the workload of MC, W_{MC} of all the MBs of the current frame
5) Estimate the sum of the workload of IDCT+VLD, $W_{VLD\_IDCT}=W- W_{VLD\_IDCT}$, where W is the workload of the frame.
6) Estimate the average number of Huffman coefficients of MB in the frame.
7) Estimate the distortion $D(i,0)$ between $P'(i,0)$ and $P(i* f_{max}/f)$.
8)Replace the dropped frame by its previous frame. For all the replacing frames, $P'(i, j), i=1\ldots M,$ estimate the distortion $D(i,j)$ between $P'(i,j)$ and $P(i* f_{max}/f+j), j=1.. f_{max}/f$.
9)Calculate $Avg(D(i,j))$ as the quality of the distortion of the target video sequence.

The details are as follows:

In Line 3, frames with different type are allocated using different ratio. We keep the ratio the same as that in *Algorithm 1*: 2:2:1 for I-, P- and B-frame.

In Line 4, we assume all MBs in I-frame are I-MBs and 1/3 MBs in P-frame are I-MBs and another 2/3 MBs are P-MBs. We also assume 1/2 MBs in B-frame are P-MB and another 1/2 MBs are B-MBs. Based on this ratio, we estimate the sum of the workload of MC of the MBs in the current frame. It should be noted that above approach is not the most accurate one. A more accurate approach can employ the residual error to estimate the number of I-, P- and B-MBs of the frame. However, our simple scheme is designed to select the best frame rate. Experimental results show that this simple approach works sufficiently well. For I-, P- and B-MB, we use a constant value to estimate the MC workload. The constant value is obtained from statistical analysis. Again, this is not the most accurate approach, but is sufficient for our purpose.

In Line 6, the number of Huffman codes is estimated using the decoding workload model proposed in Chapter 3

In Line 7, the distortion $D(i,0)$ is calculated as Equation 5.1, 5.2 and 5.4

In Line 8, the distortion $D(i,j)$ is calculated as Equation 5.5..

In the proposed scheme we have many parameters such as $W_{huff}(N)$, W_{temp}, W_{prop}, W_{mse} and W_{res}. They are all obtained from the statistical analysis: $W_{huff}(N)$ is obtained from the experiment where we select a 8*8 block from a raw picture, performing the DCT operation, setting the coefficients after position N as zero and finally calculating the difference between the original block and the block after IDCT. For W_{temp}, W_{prop}, W_{mse}

and W_{res}, we use a set of video as the training set (the same video clips we used in the experiments of Section 3.3). We enumerate the four parameters from 0~10 with a step of 0.1 and select the values with best estimation result.

5.3 Workload Control Scheme

According to the decoding workload model proposed in Chapter 3: an MPEG video is made up of a sequence of Macroblocks (MBs), and modeling the video decoding workload is decomposed to modeling three major tasks of decoding one MB: Variable Length Decoding (VLD), Inversed Discrete Cosine Transform (IDCT) and Motion Compensation (MC). The decoding workload of VLD is modeled as a linear function of the number of Huffman codes. The decoding workload of IDCT is modeled as a lookup table indexed by the last position of Huffman codes. The workload of MC is modeled as a lookup table indexed by motion vectors' precisions. And for different types of MB, the parameters of the models can be different. Thus, we can control the decoding workload by adjusting the number of Huffman codes, MB type and motion vector precision.

The workload control can work at three levels: frame level, MB level or task level (Slice level is to provide the ability of robustness, so we do not take it into consideration). For an encoder, workload control in frame and MB level is similar to the conventional bit rate control. However, in task level, rate control and workload control are significantly different. Rate control scheme only considers the quantization level of DCT coefficients, which is proportional to the bit rate. Workload control needs to consider multiple factors

and their tradeoff. For example, if we allocate more workload to the VLD or IDCT task, we can have more Huffman codes. This increases the video quality. However, allocating more workload to the VLD or IDCT task will result in allocating less workload to the MC task. This may limit the motion vectors' precision and thus decreasing the video quality. This problem becomes even harder if we also consider the MB type. In this section, as the first step of our research on the decoding-workload-aware encoder, we do not consider the task level workload control: we simply fix the workload of the MC task by fixing the MB type and motion vectors from the conventional motion estimation procedure. We only control the decoding workload of IDCT and VLD tasks by adjusting the number of Huffman codes.

We design two strategies for the frame level and MB level workload control, respectively. The strategies allocate the workload so that the encoded bitstream can have a better video quality within the constraint of decoding workload. The two strategies can be summarized as follows:

- In frame level, the workload is allocated based on statistical ratio of different frame types and the ratio is adjusted according to the recent history.
- In MB level, the workload is allocated based on the image complexity which can be estimated by the variance or MSE.

The experiment results in Section 4 show that these two strategies can improve the video quality. *Algorithm 5. 2* describes how the workload control scheme works.

Algorithm 5.2. Workload Control Scheme

1) Allocate the workload for the current GOP according to the constraint, GOP size and frame rate on an average basis.
2) For all the frames in the GOP:
3) Allocate the workload to the current frame according to the frame type and history record.
4) Run motion estimation for all the MBs of the current frame, decide their MB types, record their MSEs (or VAR for I-MB) and motion vectors.
5) Estimate the workload of MC for all the MBs based on the results from 4) using the workload model.
6) For all the MBs in the current frame:
7) Allocate the workload of current MB by its MSE/VAR and MB type.
8) From the motion vectors get in the line 4, estimate the workload of VLD+IDCT for all possible number of quantization scales using workload model. Select out the number of quantization scale having the workload closest to $W_{VLD\_IDCT} = W_{mb} - W_{MC}$, where W_{mb} is the workload allocated for the MB, and W_{MC} is the workload of MC get in line 5.
9) Encode the MB.
10) Update the status.

The details are as follows:

In line 3, the decoding workload for current frame (W_i, W_p and W_b for I-frame, P-frame and B-frame) is allocated as:

$$W_i = W / (1 + \frac{N_p X_p}{X_i K_P} + \frac{N_b X_b}{X_i K_b}) \quad (5.7)$$

$$W_p = W / (N_p + \frac{N_b K_p X_b}{X_p K_b}) \quad (5.8)$$

$$W_b = W(N_b + \frac{N_p K_b X_p}{X_b K_p}) \quad (5.9)$$

where K_p and K_b are the parameters representing the ratio between I-, P- and B-frame. In our implementation, $K_p=1.0$ and $K_b=2.0$, which are obtained empirically. It means we allocate the workload to I-, P- and B-frame according to the ratio of 2:2:1. X_i, X_p and X_b are the decoding workload for the previous I-, P- and B-frames. These three parameters are to adjust the ratio of I- P- and B-frame by the history record. W is the remaining workload of the GOP, which is updated after encoding a frame. N_p, N_b are the number of P-and B-frames in a GOP.

In line 4, we use conventional motion estimation, with which the MB type is decided by comparing the MSE or VAR of the MB with a constant threshold.

In line 6, workload of the current MB, $W_{mb}(i)$ is allocated as

$$W_{mb}(i) = (W_{frame} - \sum_{j=i}^{N} W_{MC}(j)) * MSE(i) / \sum_{j=i}^{N} MSE(j) + W_{MC}(i) \quad (5.10)$$

where W_{frame} is the remaining workload of the current frame; $MSE(i)$ is the MSE of the i^{th} MB (or $VAR(i)$, if the i^{th} MB is an I-MB); N is the number of MBs of the frame. $W_{MC}(i)$ is the workload of MC of the i^{th} MB. The rationales behind this equation are: 1) as mentioned earlier, we do not change motion vectors' precision or MB type after motion estimation, the workload of MC can be regarded as fixed; 2) we allocate more workload to VLD and IDCT tasks of the MB which has larger MSE/VAR. A MB with larger MSE/VAR implies more residual error. Therefore, it requires more Huffman codes for

the encoding. And to decode a MB with more Huffman codes, more decoding workload is required in VLD and IDCT tasks.

In line 9, we use the quantization scale obtained from line 8 for the actual encoding (generating the bitstream). If the encoder also employs a rate control scheme, we will get another quantization scale for the rate control. In this case, both the decoding workload constraint and bit rate constraint can be satisfied by selecting the larger quantization scale.

In line 10, we estimate the decoding workload using the parameters extracted from the encoded MB and update status of the scheme.

To summarize, in the frame level, we allocate the workload based on statistical ratio between different components which is updated with a moving average of recent history; in the MB level, we allocate the workload based on the image complexity. The experiment results in Section 4 show that these two strategies improve the video quality considerably.

5.4 Evaluation

5.4.1 Workload Control Scheme Evaluation

Experiment Setup

For proving of concept of the proposed decoding-workload-aware video encoding, we employ the MPEG-2 as the video format. We modify MPEG-2 reference encoder to a

decoding-workload-aware encoder. In our experiments, we select 12 raw video sequences which are shown in *Table 1*. Each of them is encoded under 11 workload constraints: 20 MHz, 30 MHz, 40 MHz, 50 MHz, 60 MHz, 80 MHz, 100 MHz, 120 MHz, 150 MHz and 200 MHz. We use MPEG-2 decoder of TCPMP project [4] as the target decoder. We use SimpleScalar [5] to simulate the decoding procedure and record the actual decoding workload, which is then compared with the workload constraints.

Experiment Results

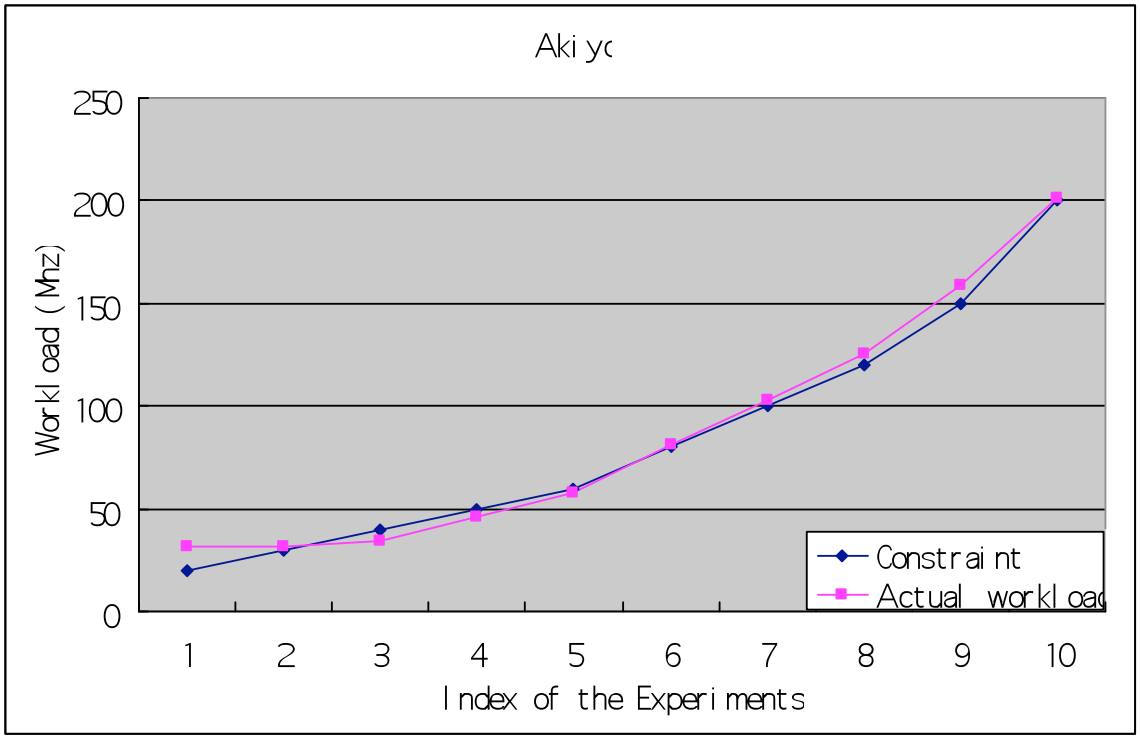

Figure 5.4 The Comparison between the constraint and actual decoding workload for sequence 'akiyo'.

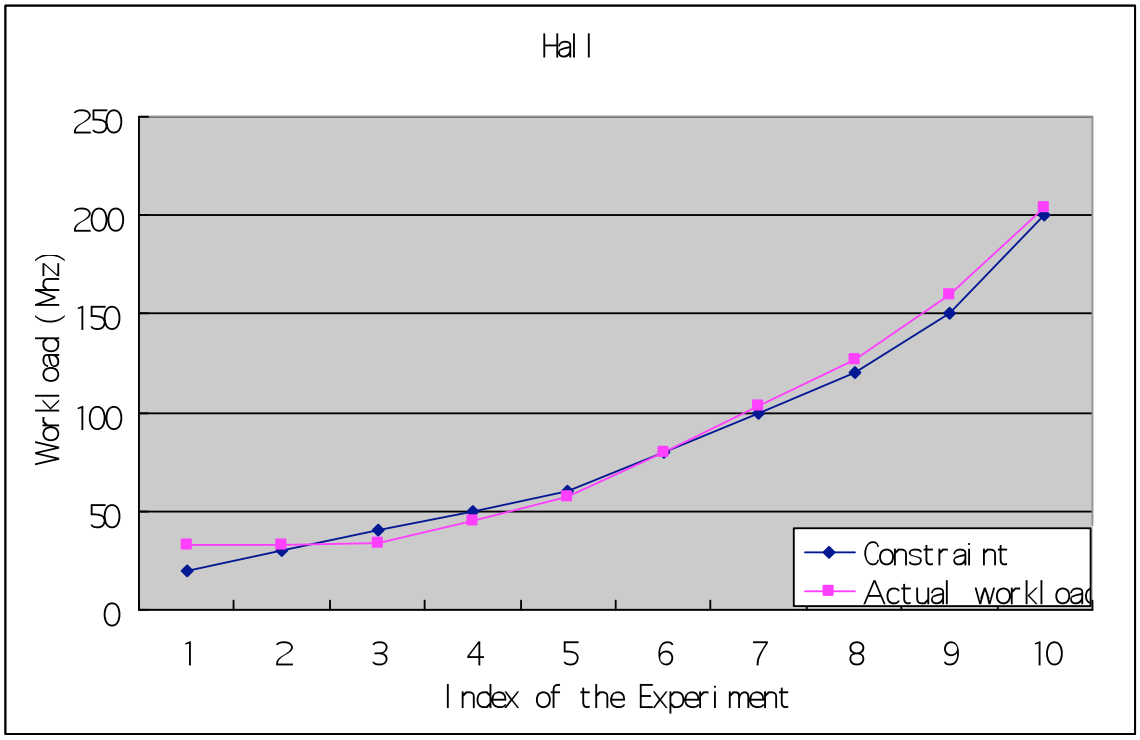

Figure 5.5 The Comparison between the constraint and actual decoding workload for sequence 'hall'.

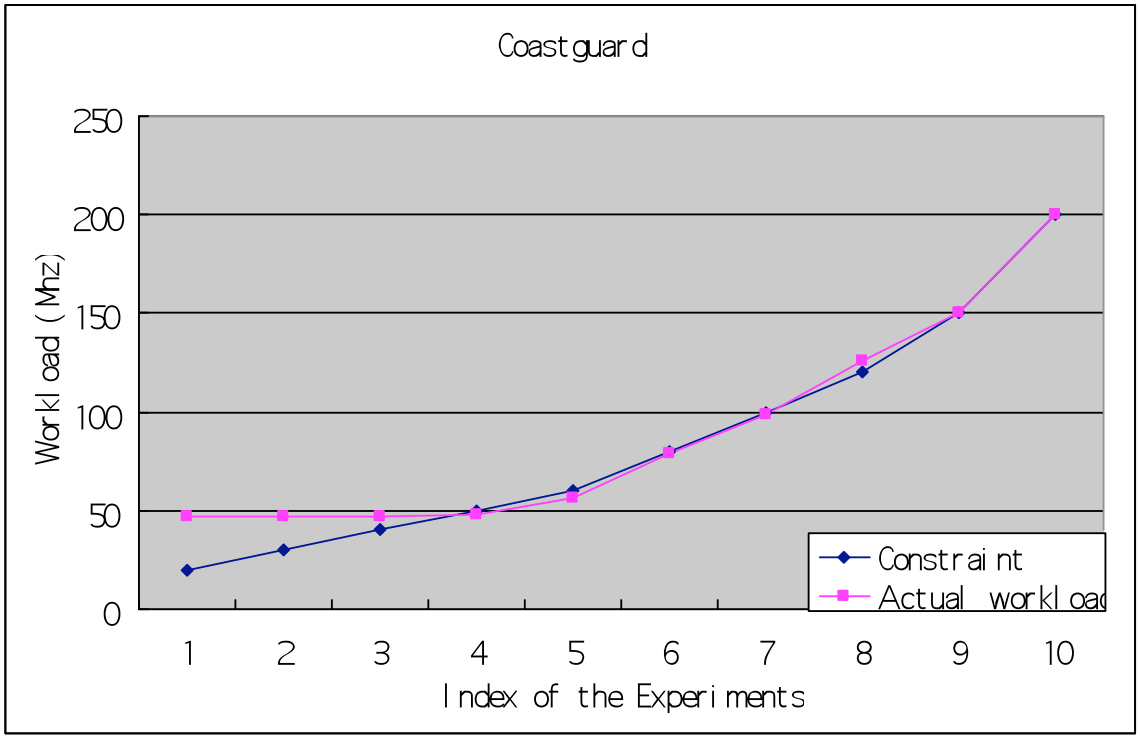

Figure 5.6 The Comparison between the constraint and actual decoding workload for sequence 'coastguard'.

Figure 5.4~Figure 5.6 show the comparison between the constraint and the actual decoding workload for the sequences *akiyo, hall and coastguard*. The results of the other sequences also show the similar matching. Two curves in the figures are the constraint and the actual decoding workload, respectively. It can be observed that, in most cases, the actual workload is very close to the constraint. However, when the constraint is very low, the actual decoding workload is beyond the constraint. It is because each sequence has a minimum decoding workload requirement which is dependent on the video content. When the motion of the video is large, the residual error in P- and B-frame is large, which demand more decoding workload. For example, *akiyo*'s motion is smaller than *hall*'s, which again smaller than *coastguard*'s. Thus, in the figures, we can observer the minimum decoding workload requirement of *akiyo* is around 32 MHz, which is smaller than *hall*'s 34 MHz, which again smaller than *coastguard*'s 47 MHz. Thus the minimum decoding workload requirement will be large, and vice visa. The average difference between the constraint and actual decoding workload is less than 1.8 %. That indicates the workload control scheme controls the decoding workload very well.

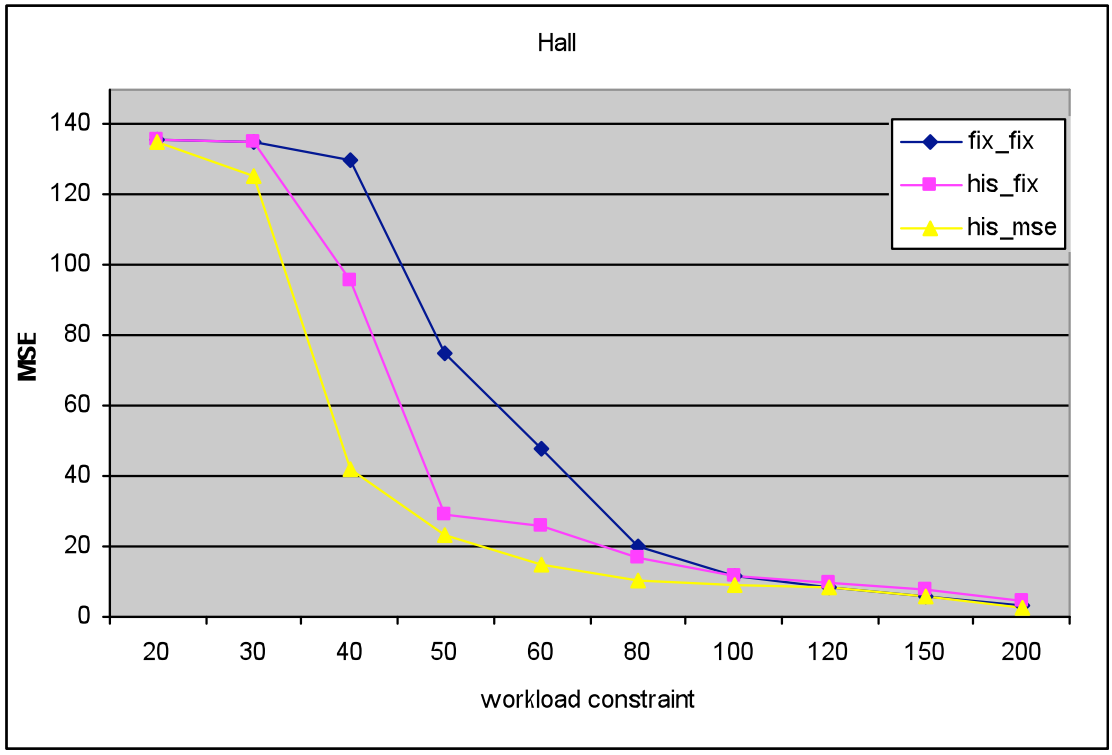

Figure 5.7 The Comparison between the video distortions between different workload control schemes for the sequence 'hall.

Next, we evaluate the strategies we employ in the workload control scheme. In *Figure 5.7* we compare the video quality between the bitstream generated by our scheme and the bitstreams generated by the scheme without the strategies for the video sequence *hall*. The x-axis represents the workload constraint and the y-axis represents the MSE of the encoded video bitstream. The results of the other sequences are similar. In the figure, the curve *his_mse* represents MSE of the bitstream generated by the scheme using two strategies. The curve *his_fix* represents the MSE of the bitstream generated by the scheme only using the strategy in frame level. In the MB level, we allocate workload according to a fixed ratio. And the curve *fix_fix* represents the MSE of the bitstream generated by the scheme using no strategy. We allocate workload based on fixed ratios in both frame and MB level. It is observed that, under the same constraint, the bitstream generated by using

two strategies has better quality than the bitstream generated by using only one strategy, which again better than the bitstream generated by using no strategy. It implies that both two strategies: 1) allocating workload in frame level based on history record and, 2) allocating workload in MB level based on MSE efficiently work in the workload control scheme.

5.4.2 Frame Rate Selector Scheme Evaluation

Experiment Setup

In the experiment, given a picture sequence and the workload constraint, we select the best frame rate from the candidates using the frame rate selection scheme. To evaluate the result, we encode and decode the sequence under the same workload constraint for all the frame rate candidates. After that, the dropped frames are replaced with the previous un-dropped frames and the average MSE is calculated. We check if the frame rate selected by our scheme has the smallest MSE. In the experiment, we use 12 different picture sequences and 14 workload constraints: 10, 20, 30, 40, 50, 60, 80, 100, 120, 150, 180, 200, 250 and 300 MHz; and frame rate candidates are 5, 10, 15, 20 and 25 fps.

Experiment Results

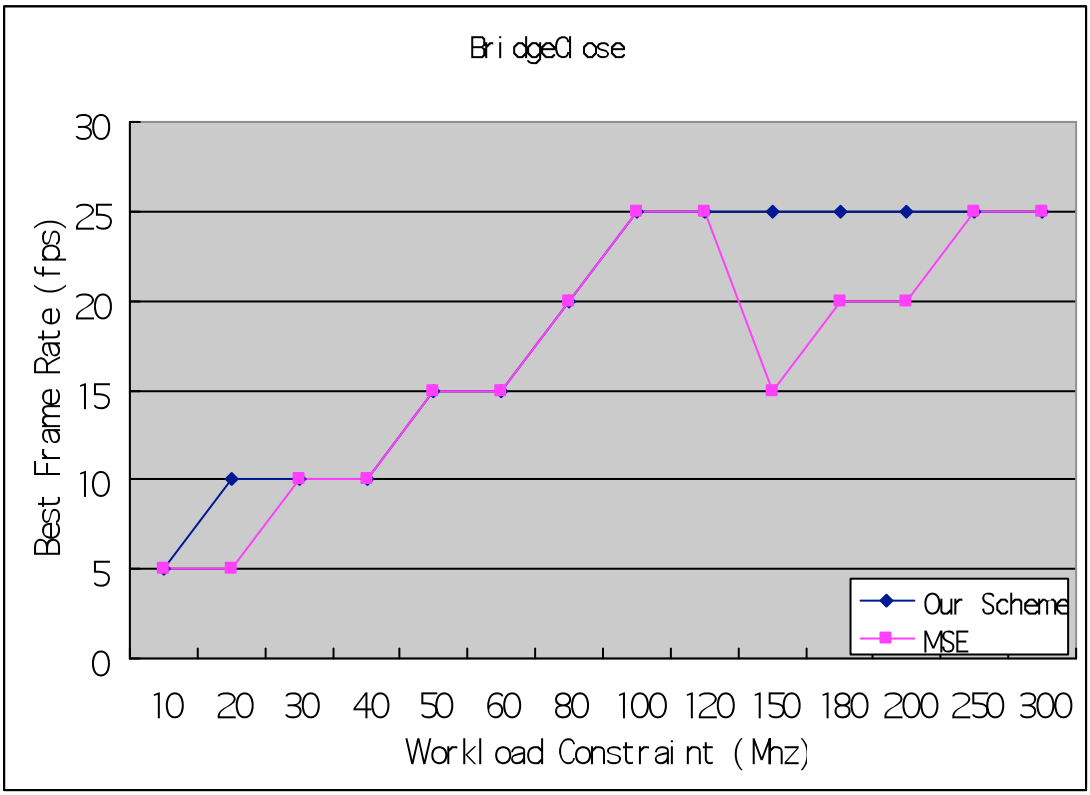

Figure 5.8 The Comparison between our scheme and MSE for the sequence 'bridgeclose'

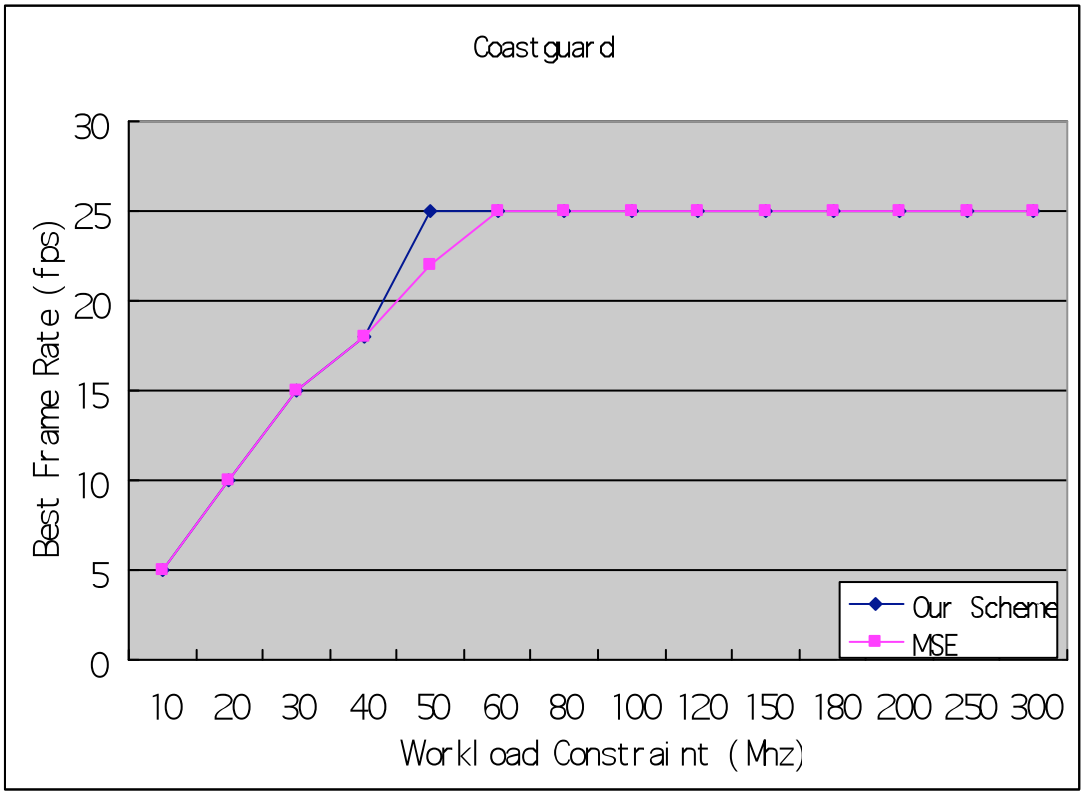

Figure 5.9 The Comparison between our scheme and MSE for the sequence 'coastguard

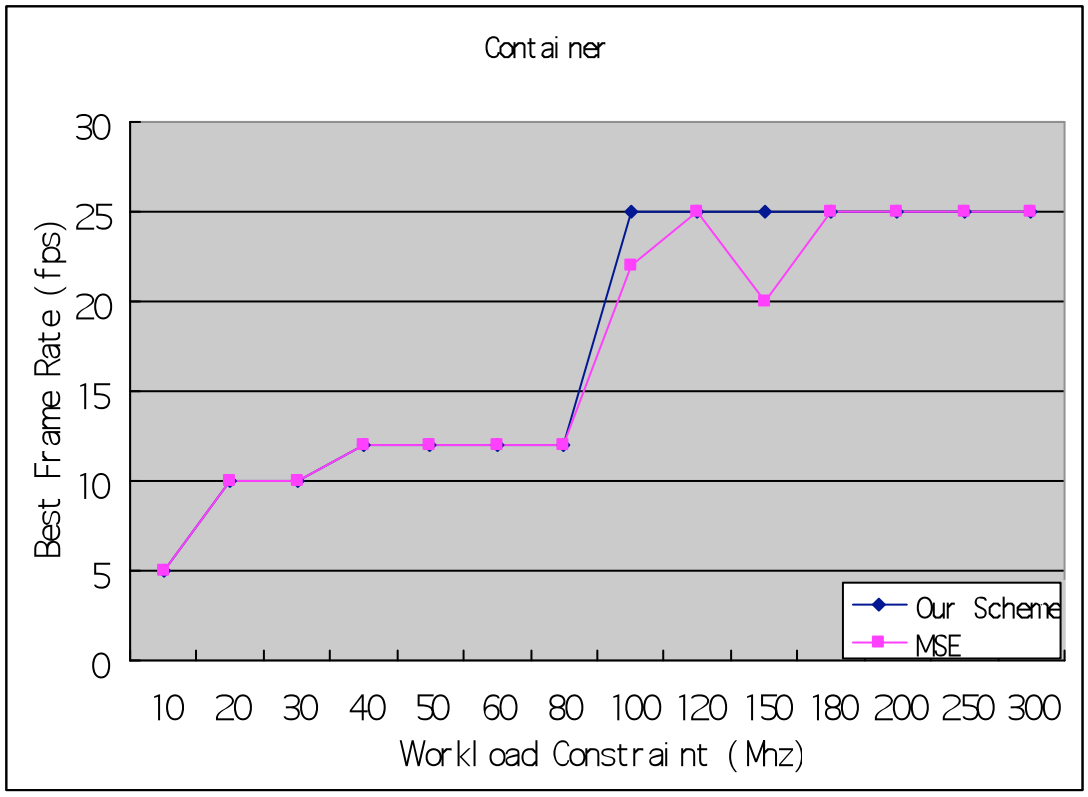

Figure 5.10 The Comparison between our scheme and MSE for the sequence 'container

Figure 5.8~Figure 5.10 show the comparison between our scheme and MSE for the sequence 'bridgeclose', 'coastguard' and 'container'. The results for the other sequences are similar. It is observed that our scheme and MSE matches well in most cases. The percentage that the frame rate selected by our scheme has the smallest MSE value is 74.4%. The percentage that the frame rate selected by our scheme has the smallest or second smallest MSE value is 90.4%. Furthermore, the cases our scheme does not match the MSE, are possibly because MSE does not reflect the video quality accurately. For example in Figure 5.8, when the workload constraint is 100 and 120 MHz, MSE selects the best frame as 25fps. However, when the workload constraint increases to 150, it selects the best frame as 15fps. That does not make sense. Intuitively, as the decoding workload increase, the best frame rate should increase, at least not decrease. And in

Figure 5.10, we can also observe the similar case. In these cases, our scheme is more reasonable than MSE.

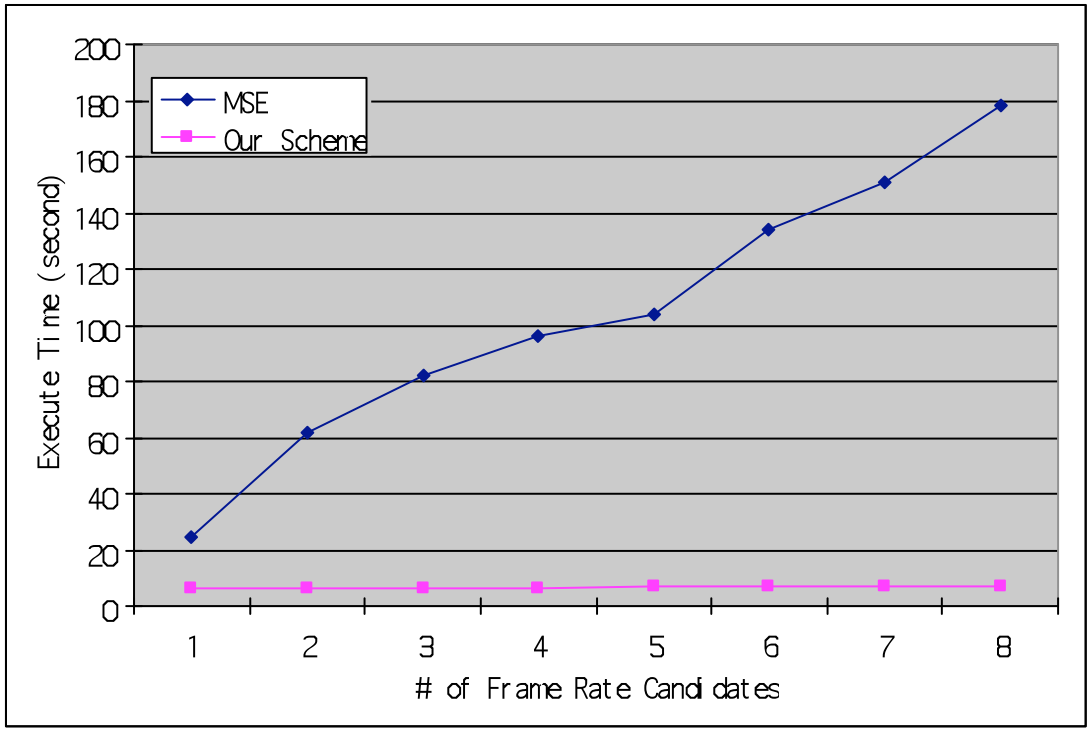

Figure 5.11 The complexity comparison between the two schemes

Compared to the conventional approach, such as MSE, our scheme has a much lower computational complexity. If we use the conventional approach, we have to encode, decode and calculate MSE for *n* times, where *n* is the number of the frame rate candidates; while in our scheme, we run the motion estimation (a part of the encoding process), calculate MSE and variance only once. A comparison of time complexity of the two schemes is shown in Figure 7. The test was run on a desktop with Pentium 4 CPU and 1G RAM running Windows XP. As shown in *Figure 5.11*, the execution time increases with the number of frame rate candidates for the conventional approach, while the execution

time for the proposed scheme is almost constant. When the number of frame rate candidates is 8, our scheme is about 25 times faster than the conventional approach.

5.5 Summary

In this chapter, we have presented a novel decoding-workload-aware video encoding scheme with two main contributions: a decoding workload control scheme and a fast frame rate selection scheme. The workload control scheme can control the decoding workload accurately when the generated video bitstream using the proposed scheme is decoded in a target client. The fast frame rate selection scheme can select out the most suitable target frame rate, balancing the spatial and temporal distortions, before the actual encoding.

We believe that the proposed fast frame rate selection scheme is not only useful for workload control but also for rate control. On the other hand, our workload control scheme still has a lot of room for improvement. For example, the workload allocation in the task level is an important and interesting problem to study in the future.

Chapter 6

Discussion and Future Works

The purpose of this thesis is to propose a scalable solution that can provide acceptable quality of service in mobile video applications yet matches the decoding workload constraints of end devices. In the thesis, we have first established a decoding workload model based on the analysis of the MPEG bitstream. Next, we have proposed a decoding workload scalable transcoder and encoder, which can produce the target video clip according to the workload constraint of the mobile device. To our best knowledge, this is the first attempt at studying the decoding workload of the mobile video application in such a comprehensive manner.

The decoding workload model is the core of the thesis. How well the transcoder and encoder can control the target decoding workload completely relies on the model's

accuracy. In this thesis, we have established the model based on detailed analysis of the MPEG video structure and different decoder implementations, which makes the model very accurate. On the same time, this approach also renders the model highly dependent on video structure and decoder's architecture. If the video structure or the decoder's architecture is not considered in the model, the model does not work anymore. That is why our model does not work on the H.264 format and performs badly when the cache is not hit. On the other hand, if we establish a model in a more abstract way, such as with the virtual decoding complexity in [16], we cannot simultaneously guarantee its accuracy and efficiency. Furthermore, it is difficult to abstract all video formats, for example, DCT based video formats and wavelet based video formats into one single model. Our further work on the decoding workload model will be: 1) we will continue improving the current model. The experimental results in Chapter 3 show that the cache mechanism has a significant impact on the model's accuracy especially when the cache miss ratio is high. We will take it into consideration. 2) We will extend the model to other video formats such as H.264 and scalable video coding [57, 72]. In fact, although current scalable video coding is designed mainly for bit rate scalability, it can also be applied for controlling decoding workload if the model can accurately predict 1) the decoding workload for both base layer and enhancement layer bitstreams; 2) the overhead for combining multiple layers.

Besides the decoding workload model, the compression domain objective quality measure is another major contribution in the thesis. The proposed measures are designed to judge the tradeoff between temporal distortion and spatial distortion. Although they are

fast and accurate, they are not yet satisfactory enough. For example, the measure proposed for the transcoder is not able to compare the quality of video clips of different frame sizes. As a result, the proposed transcoder is incapable of spatial scalability. Also, the reason why allocating decoding workload at the MB level is so difficult is because the proposed objective measure for the encoder is not able to judge the tradeoff between the number of DCT coefficients, MB type and motion compensation type. In our future work, we will study the measure more thoroughly to solve the problems mentioned above. We note that the compression domain video quality measure also highly depends on video format. For speed, the measure has to know how the video is encoded from the spatial domain data to the compression domain bits so that the video quality can be estimated in the compression domain. If we use generic video measures, such as MSE and PSNR, we have to decode the video into spatial domain, which is relatively slow. This tradeoff should be realized when we design the system.

As we have mentioned in the Introduction section, the study of decoding workload scalability of relevance to the study of energy scalability. Although we do not study energy directly in this thesis, we can still take advantage of the decoding workload model to save the client energy. In our paper [55], we have combined the decoding workload model with the idea of the dynamic voltage scaling (DVS) approach to reduce energy consumption of the processor. According to a previous study [29], energy consumption of the processor can be computed from decoding workload, which can be easily estimated via bitstream analysis using the proposed decoding workload model. Therefore, given a video clip, we can predict its energy consumption without actual decoding it. Based on

this, we have proposed a scheme: When video clips are being downloaded onto a portable device, a lightweight bitstream analysis scheme runs on the desktop computer and annotates the video clip with energy consumption information. The annotated video clips are then stored in the portable device. At runtime, energy consumption information is read out and used for dynamic voltage scaling. This scheme has two main advantages: 1) analysis and computations are done at the server side, so very little overhead will be occurred at the client device. 2) We know the energy consumption distribution of the whole video file before we make our frequency scaling decision, and we can make use of the information to efficiently reduce energy consumption without any quality degradation. In this scheme, we do not consider memory energy consumption. Although memory energy consumption is not major compared to processor energy consumption [47], it should not be ignored especially when the cache is small. In our future work, we will study this topic and extend the current decoding workload model to the decoding energy model.

References

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

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

[3] 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.

[4] R. Apteker, J.A. Fisher, V.S. Kisimov, and H. Neishlos. "Acceptability and frame rate", IEEE Transactions On Multimedia, pages 32–40, 1995.

[5] T. Austin, E. Larson, and D. Ernst. "(simplescalar): An infrastructure for computer system modeling", IEEE Computer, pages 59–67, 2002.

[6] A. Bavier, B. Montz, and L. Peterson. "Predicting mpeg execution times. ACM SIGMETRICS Performance Evaluation Review", 1998.

[7] 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.

[8] K. Choi., K. Dantu, W. Cheng, and M. Pedram. "Frame-based dynamic voltage and frequency scaling for a mpeg decoder", ICCAD, pages 732–737, Nov 2002.

[9] Q. Wu, P. Juang, M. Martonosi, and D. W. Clark, "Formal online methods for voltage/frequency control in multiple clock domain microprocessors", ASPLOS, 2004.

[10] G. Heising and M. Wollborn. “MPEG-4 version 2 video reference software package, acts ac098 mobile multimedia system (momusys)”, IEEE Transactions On Consumer Electronics, Dec 1998.

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

[12] M. Mattavelli and S. Brunetton. “Implementing real-time video decoding on multimedia processors by complexity prediction techniques”, IEEE Transactions On Consumer Electronics, pages 760–767, Aug 1998.

[13] M. Mattavelli and S. Brunetton. “Implementing real-time video decoding on multimedia processors by complexity prediction techniques”, IEEE Transactions On Consumer Electronics, pages 760–767, Aug 1998.

[14] K. Ngan, T. Meier, and Z. Cheng. “Improved single-video object rate control for MPEG-4”, IEEE CSVT, pages 760–767, May 2003.

[15] W. Pan and A. Ortega. “Complexity-scalable transform coding using variable complexity algorithm”, Data Compression Conference, pages 263 – 272, Mar 2000.

[16] M. Schaar and Y. Andreopoulos. “Rate-distortion-complexity modeling for network and receiver aware adaptation”, IEEE Transactions On Multimedia, pages 471–479, Jun 2005.

[17] D. Son, C. Yu, and H. Kim. “Dynamic voltage scaling on mpeg decoding. ICPADS”, pages 633–640, 2001.

[18] 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.

[19] G.M. Wilson. “Psycho physiological indicators of the impact of media quality on users”, Proceeding on HCI, pages 95–96, Mar 2001.

[20] “MPEG-1: Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbits/sec, Part 2: Video”, ISO/IEC JTC1/SC29/WG11, 11172-2,1993

[21] “MPEG-2: Generic coding of moving pictures and associated audio information: Video”, ISO/IEC JTC1/SC29/WG11, 13818-2, 2000

[22] “MPEG-4 visual finial draft international standard, part 2: Visual”, ISO/IEC JTC1/SC29/WG11 14496, 2003

[23] “MPEG-4 visual finial draft international standard, part 10: Advanced Video Codec”, ISO/IEC JTC1/SC29/WG11 14496-10 Oct.2003

[24] “ITU-T Recommendation H.261, video codec for audiovisual services at p x 64 kbit/s”, 1990

[25] “ITU-T Recommendation H.263, video coding for low bitrate communication”, ver. 1, Nov 1995

[26] J. Sullivan, T. Wiegand, “Rate-Distortion Optimization for Video Compression”, IEEE Signal Processing Magazine, 1998

[27] Z. He, Y. Liang, L. Chen, D. Wu, “Power-Rate-Distortion Analysis for Wireless Video Communication under Energy Constraints”, IEEE Transactions On Circuit and System for video Technology, Vol. 15, Issue 5, pp. 645-658, May. 2005.

[28] K. Ngan, T. Meier, Z. Cheng, “Improved Single-video Object Rate Control for MPEG-4”, IEEE CSVT, May 2003.

[29] P. Anantha, W. Robert, “Minimizing Power Consumption in Digital CMOS Circuits”, Proc. of the IEEE, VOL.83, No.4. Apr. 1995

[30] T. Sakurai, A. Newton, “Alpha-power law MOSFET model and its application to CMOS inverter delay and other formulas”, IEEE Journal of Solid State Circuits, VOL.25, No. 2, pp. 584-594, Apr. 1990.

[31] J. Pouwelse, K. Langendoen, I. Lagendijk, H. Sips, “Power-aware Video Decoding”, 22nd Picture Coding Symposium, 2001.

[32] T. Simunic, L. Benini, G. De Micheli, “Energy Efficient Design of Portable Wireless Devices”, International Symposium on Low Power Electronics and Design, pp.49-54, 2000

[33] T. Simunic, L. Benini, G. De Micheli, “Dynamic Power Management for Portable Systems”, the 6th Internal Conference on Mobile Computing and Networking, pp. 22-32, 2000

[34] Y. Liu, A. Maxiaguine, S. Chakraborty, W. Ooi, “Processor Frequency Selection for SoC Platforms for Multimedia Applications,” Real-Time Systems Symposium, 2004, pp.336 – 345, Dec. 2004.

[35] M. Pinson, S. Wolf, “Comparing Subject Video Quality Testing Methodologies”, Proceedings of SPIE, 2003.

[36] T. Pering, T. Burd, R. Brodersen, “The Simulation and Evaluation of Dynamic Voltage Scaling Algorithm”, ISLPED 98, pp. 76 – 81, Aug. 1998.

[37] Z. Lu, J. Lach, M. Stan, “Reducing Multimedia Decode Power using Feedback Control”, ICCD 2003, pp. 489 – 496，, Oct. 2003.

[38] C. Im, Y. Kim, S. Ha, “Dynamic Voltage Scheduling Technique for Low-Power Multimedia Applications Using Buffers”, ISLPED 2001, pp.34 – 39, Aug. 2001.

[39] W. Yuan, K. Nahrstedt, “Practical Voltage Scaling for Mobile Multimedia Devices”, ACM MM 2004, pp. 924 – 931, Oct .2004.

[40] K. Choi, R. Soma, M. Pedram, “Off-chip Latency-Driven Dynamic Voltage and Frequency Scaling for an MPEG Decoding”, DAC 2004, pp. 544 – 549, Jun. 2004.

[41] H. Shu, L. Chau, “An efficient arbitrary downsizing algorithm for video transcoding”, Circuits and Systems for Video Technology, IEEE Transactions on, pp. 887- 891, Jun, 2004.

[42] Y. Liang, L. Chau, Y. Tan, “Arbitrary downsizing video transcoding using fast motion vector reestimation”, Signal Processing Letters, IEEE, pp. 352 – 355, Nov, 2002.

[43] C. Surendar, V. Adim, “Application-specific Network Management for Energy-Aware Streaming of Popular Multimedia Formats”, Proceedings of the General Track of the annual conference on USENIX Annual Technical Conference, pp. 329-342, 2002

[44] M. Kienzle, P. Shenoy “WirelessNetwork Interface Energy Consumption Implications of Popular Streaming Formats”, The International Society of Optical Engineering, pp. 85-99, Jan, 2002.

[45] J. Korhonen, Y. Wang, “Power-Efficient Streaming for Mobile Terminals”, Proceedings of the international workshop on Network and operating systems support for digital audio and video, pp. 39-44, 2005.

[46] Y. Lu, G. Micheli, “Comparing System-Level Power Management Policies”, IEEE Design and Test of Computer, vol. 18, pp. 10-19, Mar. 2001.

[47] J.R. Lorch, A.J. Smith, “Software Strategies for Portable Computer Energy Management”, IEEE Personal Communications Magazine, 1998.

[48] A. Iranli, W. Lee, M. Pedram, “Backlight Dimming in Power-Aware Mobile Displays”, Proceedings of Design Automation Conference, pp. 604-607, 2006.

[49] A. Iranli, M. Perdram, “DTM: Dynamic Tone Mapping for Backlight Scaling”, Proceedings of Design Automation Conference, pp. 612-616, Jun. 2005

[50] I. Choi, H.Shim, N. Chang, “Low-power color TFT LCD display for hand-held embedded systems”, Proceedings of Internal Symposium on Low Power Electronics and Design, pp. 112-117, Aug. 2002.

[51] N. Chang; I. Choi, H. Shim, “DLS: Dynamic Backlight Luminance Scaling on Liquid Crystal Display”, IEEE Transactions on VLSI Systems, pp. 837-846, Aug. 2006.

[52] W. Cheng, M. Pedram, “Power Minimization in a Backlit TFT-LCD by Concurrent Brightness and Contrast Scaling”, IEEE Transactions on Consumer Electronics, pp. 25-32, Feb. 2004.

[53] H. Shim, N. Chang, M. Pedram, “A Compressed Frame Buffer to Reduce Display Power Consumption in Mobile Systems”, Proceedings of the ASP-DAC 2004. Asia and South Pacific, pp. 819-824, Jan. 2004.

[54] W. Yuan, K. Nahrstedt, “Energy-Efficient Soft Real-Time CPU Scheduling for Mobile Multimedia Systems”, Proceedings of the nineteenth ACM symposium on Operating systems principles, pp. 149-163. Oct. 2003.

[55] Y. Huang, S. Chakraborty, Y. Wang, “Using Offline Bitstream Analysis for Power-Aware Video Decoding in Portable Devices”, Proceedings of the 13th annual ACM international conference on Multimedia, pp. 299-302, Nov. 2005.

[56] T. Berger, “Rate Distortion Theory”, Prentice Hall, Englewood Cliffs, NJ, 1984.

[57] M. Gallant, F. Kossetini, “Efficient scalable DCT-based video coding at low bit rates”, ICIP 99, Vol. 3, pp. 782-786, Oct. 1999.

[58] M. Bystrom, W. Modestino, “Combined Source-Channel Coding Schemes for Video Transmission over an Additive White Gaussian Noise Channel”, IEEE Journal on selected areas in communications, Vol.18, No.6, pp. 880-890, Jun. 2000.

[59] D. Wu, T. Hou, W. Zhu, Y.-Q. Zhang, “An End-to End approach ofr Optimal Mode Selection in Internet Video Communication, Theory and Application”, IEEE Journal on selected areas in communications, Vol.18, No.6, Jun. 2000.

[60] Z.He, S.K. Mitra, “A Unified Rate-Distortion Analysis Framework for Transform Coding”, IEEE Transactions on Circuits and System for Video Technology, Vol. 11, No. 12, Dec. 2001.

[61] H. Schwarz, D. Marpe, “Overview of the Scalable Video Coding Extension of the H.264/AVC Standard”, IEEE Transactions On Circuits and Systems for Video Technology, pp. 1103-1120, Sep. 2007

[62] C. Poellabauer, K. Schwan, “Energy-Aware Media Transcoding in Wireless Systems”, Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications, pp. 135-144, Mar. 2004.

[63] R. Han, C. Lin, R. Smith, B. Tseng, V. Ha, “Universal Tuner: A Video Streaming System for CPU/Power-Constrained Mobile Devices”, Proceedings of the 9th ACM International Conference on Multimedia 2001, pp.632-633, Sep. 2001.

[64] K. Barr, K. Asanovic, “Energy Aware Lossless Data Compression”, In Proc. of the First International Conference on Mobile Systems, Applications and Services. Pp. 250 – 291, May, 2003.

[65] P. Pakdeepaiboonpol, S. Kittitornkun, “Energy Optimization for mobile MPEG-4 Video Decoder”, 2005 2nd International Conference on Applications and Systems, Jan, 2006.

[66] J. Flinn, M. Satyanarayanan, “Energy-aware adaption for mobile application”, ACM SIGOPS Operating Systems Review, pp. 48-63, Dec. 1999.

[67] Z. He, Y. Liang, L. Chen, I. Ahmad, D. Wu, “Power-Rate-Distortion Analysis for Wireless Video Communication Under Energy Constraints”, IEEE Transactions on circuits and systems for video technology, May. 2005.

[68] J. Paradiso, T. Starner, “Energy scavenging for mobile and wireless electronics”, IEEE Pervasive, pp. 18-27, 2005.

[69] “Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to About 1.5 Mbit/s – Part 2: Video, ISO/IEC 11172-2 (MPEG-1 Video)”, ISO/IEC JTC 1, Mar. 1993.

[70] “Generic Coding of Moving Pictures and Associated Audio Information – Part 2: Video”, ITU-T Rec. H.262 and ISO/IEC 13818-2 (MPEG-2 Video), ITU and ISO/IEC JTC 1, Nov. 1994.

[71] "Coding of audio-visual objects – Part 2: Visual", ISO/IEC 14492-2 (MPEG-4 Visual), ISO/IEC JTC 1, Version 1: Apr. 1999, Version 2: Feb 2000,, Version 3: May 2004.

[72] H. Schwarz, D. Marpe, T. Wiegand, "Overview of the Scalable Video Coding Extension of the H.264/AVC Standard", IEEE Transactions on Circuits and System for Video Technology, Sep. 2007.

The Publications during the PhD Study

[1] Yicheng Huang, Guangming Hong, Vu An Tran, Ye Wang: Decoding-workload-aware video encoding. NOSSDAV 2008: 45-50

[2] Yicheng Huang, Samarjit Chakraborty, Ye Wang: Watermarking Video Clips with Workload Information for DVS. VLSI Design 2008: 712-717

[3] Yicheng Huang, An Vu Tran, Ye Wang: A compressed domain distortion measure for fast video transcoding. ACM Multimedia 2007: 787-790

[4] Yicheng Huang, An Vu Tran, Ye Wang: A workload prediction model for decoding mpeg video and its application to workload-scalable transcoding. ACM Multimedia 2007: 952-961

[5] Jari Korhonen, Yicheng Huang, Ye Wang: Generic forward error correction of short frames for IP streaming applications. Multimedia Tools Appl. 29(3): 305-323 (2006)

[6] Yicheng Huang, Samarjit Chakraborty, Ye Wang: Using offline bitstream analysis for power-aware video decoding in portable devices. ACM Multimedia 2005: 299-302

[7] Yicheng Huang, Jari Korhonen, Ye Wang: Optimization of source and channel coding for voice over IP. ICME 2005: 173-176