A STUDY OF SCALABILITY IN VIDEO COMPRESSION: RATE-DISTORTION ANALYSIS AND PARALLEL IMPLEMENTATION

A Thesis

Submitted to the Faculty

of

Purdue University

by

Gregory William Cook

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

December 2002

To my parents, Warren and Susan Cook; to my mother-in-law, Mary Norton; and in memory of my father-in-law, Oscar Norton.

ACKNOWLEDGMENTS

Chapter 2 grew out of a serendipitous collaboration with Dr. Josep Prades-Nebot, a professor at the Universidad Politécnica de Valencia, Valencia, Spain. Dr. Prades was a visiting scholar in the Video and Image Processing Laboratory (*VIPER* Lab) in 2001 and we both started learning about the interesting topic of rate-control roughly simultaneously. Dr. Prades brought one perspective to the video problem; I had another, complementary, perspective. Through this collaboration several ideas emerged, and one became the basis for Chapter 2. Dr. Prades carefully reviewed every equation in Chapter 2, and made many suggestions for the improvement of the manuscript.

Chapter 3 was initially supposed to be a two-month project. Little did I know at the start of the project that it was to be a full year and a half later when the full solution emerged. It is paradox of parallel processing that, inevitably, some small trivial operation in a serial sense causes huge problems in parallel implementation. During that time I had numerous conversations with Dr. Jamshed Patel, now at Oracle, and Dr. Ashfaq Khohkar, now at the University of Illinois at Chicago. In particular, Dr. Khohkar helped me over a rough spot in the theory, simply by listening to my explanation.

I am also grateful to organizations which support research at the University level. In particular, for the research in Chapter 2 I benefited from an Indiana Twenty-First Century Research and Technology Fund grant, and for the research in Chapter 3 I benefited from a Defense Advanced Research Projects Agency grant, a research assistantship at the C-SPAN Archives, and an Intel Foundation Fellowship.

I would also like to acknowledge my Doctoral Committee: Professors Edward Delp (major professor), Leah Jamieson, Susanne Hambrusch, and Jan Allebach. Excellent teachers all, and their help in my academic career cannot be underestimated.

My fellow officemates all provided inspiration to my academic undertakings, even if we didn't always stick to strictly academic topics in the office. In order of appearance, thanks goes to: Mary Comer, Lori Overturf, Frank Venezia, Ke Shen, Eduardo Asbun, Paul Salama, Sheng Liu, Cuneyt Taskiran, Eugene Lin, Jennifer Talavage, and Hyung-Cook Kim, as well as all the members of the *VIPER* Lab.

My family has supported me throughout my years at Purdue, even through the very difficult years where it didn't look like I would even finish the program. I have dedicated this document to my parents for their support throughout my academic career, which started in Honolulu, Hawaii all those years ago. My sons Matthew and Peter, and especially my wife Cynthia are my beginning and end, and I'm eternally grateful for their support.

Finally, and ultimately, all credit goes to Jesus the Christ, from whom all blessings flow. My favorite verse from the Holy Bible was particularly meaningful as I was researching and writing this dissertation: "Ask, and you shall receive. Seek, and you shall find. Knock, and the door shall be opened." *Matthew 7:7*

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

ABSTRACT

Cook, Gregory William. Ph.D., Purdue University, December, 2002. A Study of Scalability in Video Compression: Rate-Distortion Analysis and Parallel Implementation. Major Professor: Edward J. Delp.

Theoretical rate-distortion performance bounds are derived and evaluated for both layered and continuously rate scalable video compression algorithms which use a single motion-compensated prediction (MCP) loop. These bounds are derived using rate-distortion theory, and are based on an optimum mean-square error (MSE) quantizer. Consequently, the theory serves as a bound to all possible implementations of MCP scalable video coders which use MSE as a distortion measure. Parametric versions of the rate-distortion functions are derived which are based solely on the input power spectral density and accuracy of the MCP loop. The theory is applicable to scalable video coders which allow prediction drift, such as the data-partitioning and SNR-scalability schemes described in the MPEG-2 standard, as well as those with zero prediction drift such as fine granularity scalability MPEG-4. For video coders which allow prediction drift, MCP performed optimally in the encoder is shown to be a sufficient condition for stability of the decoder. Simulation of the optimal methods correspond well with the published results of actual system implementations. The theory is significant because it separates the effects of scalability from individual scalable video coder implementation artifacts, and can serve as a guide for potential increases in scalable video coder performance.

The problem inherent with any digital image or digital video system is the large amount of bandwidth required for transmission or storage. This has driven the research area of image compression to develop more complex algorithms that compress images to lower data rates with better fidelity. One approach that can be used to increase the execution speed of these complex algorithms is through the use of parallel

processing. The problem addressed here is the parallel implementation of the JPEG still image compression standard on the MasPar MP-1, a massively parallel SIMD computer. Developed here are two novel byte alignment algorithms which are used to efficiently input and output compressed data from the parallel system. Results are presented which show real-time performance is possible. Also discussed are several applications, such as motion JPEG, that can be used in multimedia systems.

1. INTRODUCTION

Scalability is a fascinating concept. If an algorithm or system is designed to be scalable, then it can be used over a wide range of operating points *without redesign.* Thus it can be a powerful and economically useful design technique. Scalability is not without cost, however. Possibilities include lowered performance across the operating points and higher design cost and system cost. For this dissertation, two situations where scalability is used in digital video are examined and the efficiency of the scalable solutions are analyzed. Digital video provides a difficult problem and a terrific opportunity to develop scalable systems. Because of the tremendous bandwidth and complexity of the the computations required, efficient solutions for video systems are worth careful study and design.

In Chapter 2, scalability is examined in the context of a scalable video codec. A rate-distortion model is developed *from first principles* in order to predict the performance bounds on a class of motion-compensated scalable video codecs. In the spirit of Shannon [1], no attempt is made to design a practical system. Here the effort is to separate the effects of scalability in motion compensated scalable video systems from the artifacts inherent in the compression systems themselves.

In Chapter 3, scalability is examined in the context of a parallel implementation. Scalable parallel algorithms are developed for a single-instruction multiple-data computer in order to perform intraframe video coding in real time. Here practicality is the key—scalable algorithms which are not also extremely efficient are useless. We discovered the actual video compression computations were straightforward; unexpectedly, the loss of efficiency in the scalable solution came from bottlenecks in acquiring video and delivering a compressed data stream in parallel.

2. RATE-DISTORTION ANALYSIS OF MOTION-COMPENSATED RATE SCALABLE VIDEO

2.1 Introduction

Because of the adaptability of the human eye, digital video is useful over a very wide range of data rates. From wireless video at 16 [kb/s] [2] to high-definition television at 19.39 [Mb/s] [3] each finds application in today's digital world. It is often true, however, the user's requirements may not match the initial video encoding data rate. For example, the user's requirements may be driven by a transmission network which possesses a widely varying bandwidth or error probability [2, 4]. Another example would be a video Internet site which produces both free, low quality video and pay-per-view high quality video. A low cost method of adapting digital video to the needs of the user is very important.

This adaptation can be accomplished in general one of three ways: by transcoding, by generating multiple independent bit streams, or using scalable video encoding. *Transcoding* involves decoding the original signal and re-encoding it at the desired data rate or quality [5, 6]. The bit stream may be fully decoded, known as pixel-domain transcoding, or partially decoded, such as DCT-domain decoding [7]. While the computational complexity is very high, the data rate may be set precisely to the required rate. The quality at that data rate is nearly the same quality as that obtained when re-encoding the original signal, assuming the initial stream is encoded at a sufficiently high data rate. Generally speaking, the quality of pixel-domain transcoding is better than a partial decoding technique, while the complexity of the former is much higher than the latter [7]. Alternatively, the bit stream may be independently encoded into two or more independent bit streams or packets. When using *simulcast*, one transmits simultaneously a high-quality bit stream and a low-quality bit stream

which are completely independent [5]. There are two main disadvantages: first, the data rate of the lower quality bit stream is set *a priori*—unless it is transcoded—and secondly, efficiency is low because the lower quality bit stream has a lot of redundant information when compared to the higher quality bit stream. A technique which seeks to overcome this limitation while still employing multiple independent streams is *multiple description coding* (see for example [8, 9]). Here multiple streams are generated which may all independently be decoded. Any of these streams may be combined to improve the overall quality of the decoded video. Consequently, adaptation is achieved by varying the number of streams which are used to create the decoded output. However, the coding efficiency is in general lower because the compression technique cannot be aggregated across all of the available data. Also, the number of streams must in general be decided *a priori.* A compromise between these two major categories is known as *scalable video.* Here the bit stream is divided into a base layer and one or more enhancement layers. The enhancement layers are dependent on the base layer and previous enhancement layers. The enhancement layer is generally scalable to the bit-level, and allows continuous scalability from the base layer data rate to the maximum enhancement layer rate, as is the case for transcoding. This technique retains the low computational complexity of transmitting simulcast video with coding efficiency which more closely follows that of the transcoding method.

We choose to study scalable video by deriving from first principles and evaluating rate-distortion functions for both layered and continuously rate scalable video compression algorithms which use a single motion-compensated prediction (MCP) loop. These functions are derived using rate-distortion theory, and are based on an optimum mean-square error (MSE) encoder/decoder (or codec). Consequently, the theory serves as a bound to all possible implementations of MCP scalable video codecs which use MSE as a distortion measure. By specifying translatory motion, it is possible to derive closed-form versions of the rate-distortion functions. We also derive a sufficient condition for stability when coding below the MCP rate. Further, using these bounds

we show that, for systems which deliberately employ prediction drift, an optimum base rate may be found.

Scalability, when applied to video, is the capability of decoding a compressed sequence at different data rates to achieve a desired quality. These qualities are generally grouped into three categories: signal-to-noise ratio (SNR), spatial resolution, and temporal resolution. SNR is a measure of how closely the reconstructed frame matches the original; in other words, it is a measure of the quantization noise introduced in the sequence. Spatial resolution is simply a measure of how large the image frame is. Temporal resolution is a measure of how close in time the frames of the video sequence are. The selection of which spatial and temporal resolution is driven by the user's perception and limits of acceptability. SNR is a more objective measure, and can be more easily modeled in a mathematical framework.

Scalable video compression schemes are also distinguished by how the different rates are achieved. Several scalable video coding schemes have been proposed such as Layered Scalable (LS) codecs [5], Fine-Grained Scalable (FGS) codecs [10] and an extension termed Fully Fine-Grained Scalable (FFGS) codecs [11]. In LS codecs, for example the SNR-scalability mode described in MPEG-2, the bit stream is divided into a *base layer*, that provides a minimum level of quality, and one or more *enhancement layers* that allow improvement of the quality provided by the base layer. However, the number of layers in LS codecs is usually too small to achieve a good adaptation to the continuous changes in the available bandwidth of best effort networks and the characteristics of these layers must be set *a priori*. FGS codecs (*e.g.*, MPEG-4 FGS [10, 12] and [13]) address these problems by allowing decoding of the bit stream for a very large set of different rates. In these codecs, once the minimum data rate ($R_{\min}$) and the maximum data rate ($R_{\max}$) of the service have been established, video can be delivered at almost any rate in the interval $(R_{\min}, R_{\max})$. Similar to LS codecs, the bit stream is divided into a base layer and one or more enhancement layers. The base layer is generated by a non-scalable encoder operating at R_b ($R_b \leq R_{\min}$). Fine-grained scalability can be achieved by using bit plane [10] or

embedded encoding [14, 15] of the transform coefficients in the enhancement layers. The base layer along with the enhancement layers generate a bit stream operating at $R_{\max}$. The data rate can be then decreased by the server, by intelligent routers, or by the decoder, which allows the bit stream to be adapted to the local network state or receiver characteristics. Finally, in FFGS the maximum degree of scalability is provided because embedded coding is used in both the base and enhancement layers [16, 17, 18, 4, 19, 20]. The decoding rate in this case can be even lower than $R_{\min}$, which allows a greater degree of freedom for tailoring the scalability of the encoded bit stream to the application [11].

For the purposes of illustrating a practical FFGS system, we will use a fully rate scalable wavelet codec known as *SAMCoW* (*S*calable *A*daptive *M*otion *Co*mpensated *W*avelet) [21, 17, 22, 23, 24, 25, 26, 27]. *SAMCoW*'s two main features are: (i) a modified zerotree wavelet image compression scheme, known as Color Embedded Zerotree Wavelet (*CEZW*) [28, 29], used for coding intracoded frames and predictive error frames; and (ii) adaptive block-based motion compensation [30, 16] used in the spatial domain to reduce temporal redundancy. *CEZW* is a technique that uses a combination of a unique spatial orientation tree and color transform to exploit redundancy across color components, and it has the property of being fully rate scalable. Further compression is achieved by using arithmetic coding. Adaptive motion compensation is used to overcome predicted error frame degradation when the decoder is used at different data rates. In *SAMCoW* this is accomplished by using motion estimation/compensation only at the lowest reference rate. Thus any errors in the bit stream above the reference rate do not affect motion compensation in the rest of the bit stream. Operation of *SAMCoW* in its various modes, including decoding both above and below the reference rate, are described below.

Motion-compensated prediction (MCP) is used in video compression to reduce or eliminate redundant temporal information. A block diagram of a general MCP video encoder and decoder is shown in Fig. 2.1, where in this case embedded encoders are used for the compression technique. The operation of the MCP system in a standard

(non-scalable) way is described first. In this case all of the encoding and decoding rates shown in Fig. 2.1 are equal, *i.e.*, $R_a = R_b = R_c = R_d$. First, the system is "primed" by sending a non-MCP frame known as an intracoded frame or an I-frame. In the next phase the previously encoded frame is decoded. This frame is then compared to the current frame and sections—normally 8×8 blocks—are translated to find sections in the current frame with the best match, an operation known as motion estimation. Motion compensation consists of taking these translations—known as motion vectors—and computing a frame known as the predicted frame. Subtracting the predicted frame from the current frame generates a frame known as the predicted error frame or P-frame. This frame is then encoded and transmitted to the decoder, and is also used to generate the reference frame for the next encoded frame. The cycle continues until a new I-frame is sent. The section in Fig. 2.1 which performs these tasks will be referred to as the *MCP loop* in the following discussion. Note that there are *two* data paths: a path which consists of encoded I-frames and P-frames and another which consists of the encoded motion vectors. In practice these two data paths are interleaved in the bit stream. Once the encoded data reaches the decoder the motion compensation loop is reversed. First a reference frame is generated by decoding the incoming frame—exactly as in the encoder—and adding that to the previous frame the motion compensation is applied from the decoded motion vectors to the reference frame to form the new predicted frame. This predicted frame, having been delayed by one frame period, is then added to the current P-frame to generate the reconstructed frame. Note in this case the reference frame in the decoder is identical to the reference frame in the encoder MCP loop.

For video codecs with fine-grained scalability, such as MPEG-4 FGS and *SAM-CoW*, some modifications are made to the above description. First, $R_a > R_b$, where R_a is the total system rate and R_b is known as the base layer rate. Second, in the decoder $R_c = R_b$ and R_d is set by the user requirements or the transmission system, where $R_b \leq R_d \leq R_a$. Since the reference frame in the decoder is always generated from data encoded at R_b, it is exactly equal to the reference frame in the

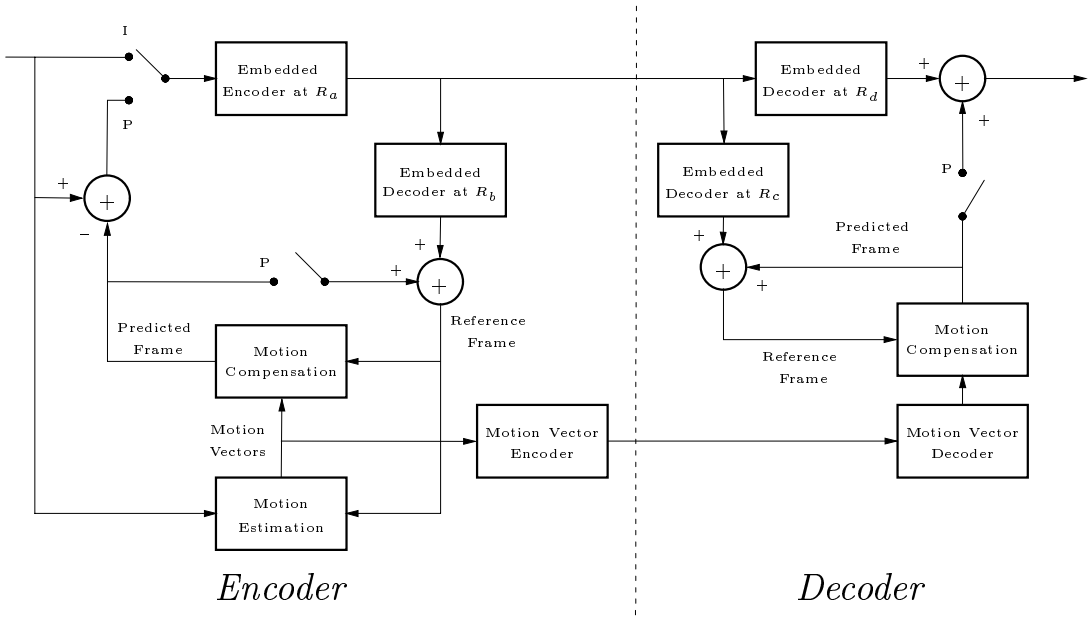


Fig. 2.1. Block diagram of a general video codec using MCP.

encoder. However, the system is now scalable because R_d is allowed to range from R_b to R_a without disruption to the MCP loop.

In this codec, however, another mode is available for decoding below the base rate. If the encoded predicted error frame is limited in rate to less than the base rate, *i.e.*, $R_c = R_d < R_b$, decoding is possible, but the reference frame in the decoder will not be identical to the reference frame in the encoder. As a consequence, the decoded frames will be produced with increased distortion compared to the optimum at that data rate [31]. This is known as *prediction drift*, and it can be very detrimental to the overall quality of the video [5, 32]. As noted, prediction drift can be avoided entirely in scheme such as FGS [10] (as long as $R_d > R_b$) since the base layer is independent of the enhanced layer. Unfortunately, this causes an inefficiency in the encoder since enhanced layer data is not included in the MCP loop. In an FFGS scheme such as *SAMCoW* [21, 17], prediction drift occurs when decoding the below the base rate, but no drift occurs above the base rate [11].

Clearly, the rate at which MCP is performed in the encoder (R_b in the above example) is critical. Optimally, all the frame information should be used in the MCP loop at each rate. Since this is not possible in a scalable system—generally speaking, the optimal motion vectors change for each rate—the critical rate is the one at which the MCP is performed. For this discussion, the *MCP rate* is defined to be the rate corresponding to the data which is used for MCP, *i.e.,* the minimum rate for which the reference frame in the encoder and decoder are identical, excluding the motion vector rate. In other papers is is also known as the "feedback rate" [20].

There is a wide body of literature on rate-distortion optimizations for video coding (*e.g.,* [33, 34, 35]) and as it applies to scalable video coding (*e.g.,* [36, 37, 38]). However, the rate-distortion functions are all *operational*; they are derived from implementations of the various encoding models using experimental data. Because of the independence of the intraframe encoding method we are able to separate the effects of scalability from the artifacts generated by an individual compression method, and thus predict performance without needing to specify an implementation.

In Section 2.2, we introduce rate-distortion theory fundamentals and the optimum MSE codec, derive some basic signal processing properties of optimum MSE codecs, and examine layered and cascaded optimum MSE codecs. We also give a summary of the results derived in [39] for a MCP video codec which employs an optimum MSE codec. Section 2.3 describes the parameterized rate-distortion functions for MCP scalable video codecs. In Section 2.4 the rate-distortion functions derived in Section 2.3 are approximated by assuming the motion is completely translatory, and a sufficient condition for stable decoding with prediction drift is obtained. The results are then used to find the optimal base rate for codecs which operate both above and below the MCP rate in Section 2.5. In Section 2.6 we assume an input power spectral density and generate the rate-distortion functions through numerical simulation. These are then compared to experimental results published in the literature in Section 2.8.

2.2 Background

Our objective is to find the general rate-distortion function of MCP scalable video compression. We start with an optimum mean square error codec for a Gaussian image model. This allows the theory to be independent of the specific implementation of the video codec, including both the spatial transform and the entropy encoding method [39]. Next, rate-distortion functions for codecs composed of two optimum MSE codecs are derived. Although their rate-distortion function is equivalent to a single optimum MSE codec, they are used extensively in Section 2.3 to simplify the derivations. Finally, in this section the rate-distortion behavior exhibited by non-scalable motion-compensating prediction hybrid coding [39] is presented. We note the rate-distortion functions derived below can be interpreted as an upper bound for non-Gaussian sources which have the same power spectral density.

2.2.1 An Introduction to Rate-Distortion

In this document we are examining the effects of compressing a signal, *i.e.,* we are interested in the differences between the input signal (uncompressed) and the output signal (after compression) when an encoder/decoder system is used. The actual workings of the encoder and decoder, while important for practical implementations, are not important here because we wish to have the *best* possible result given the constraints of a certain fidelity of the signal and limitations in transmission bandwidth. Since we are mathematically assured of the best result, all practical implementations of codecs—be they DCT, wavelet, vector quantizers, *etc.*—may be thought of as asymptotically approaching the codec described here. Thus the results obtained are valid for any of the practical implementations.

Shannon [1] pioneered the way to measure the performance of lossy compression using the method of rate-distortion analysis. Since the codec introduces distortion in the output signal, a measure of the performance of the codec is the amount of this distortion in the output signal. Many distortion measures have been proposed for

image and video signals; in this document we will use the classic mean square error method (described below).

Another measure of the performance of the codec is the rate. The rate is the minimum number of bits needed to represent the signal given a fixed distortion. Shannon showed, given a fixed distortion, the rate is the average mutual information per symbol between the input and output of the codec [1].

The two measures, rate and distortion, can then be used to describe the performance of any codec. One then can construct rate-distortion functions to characterize the codec.

Let the input to the codec be X_t which is a strictly stationary discrete-time, continuous amplitude random process. Let Y_t be the output of the codec and assume it is also strictly stationary and a discrete-time, continuous amplitude random process. Let $p(\mathbf{x}) = p(x_1, \ldots, x_n)$ be the joint probability density of n outcomes of X_t. Also, let $\rho_n(\mathbf{x}, \mathbf{y}) = n^{-1} \sum_{i=1}^{n} \rho(x_i, y_i)$ be the distortion measure which compares n successive inputs to n successive outputs. Consider all the conditional probability densities of n output Y_t given n input X_t, designated $q(\mathbf{y}|\mathbf{x})$, where $\mathbf{y} = (y_1, \ldots, y_n)$ and $\mathbf{x} = (x_1, \ldots, x_n)$. We first define the distortion as the expectation of the distortion measure [40]:

$$d(q) \triangleq \iint p(\mathbf{x}) q(\mathbf{y}|\mathbf{x}) \rho_n(\mathbf{x}, \mathbf{y}) \, d\mathbf{x} \, d\mathbf{y}. \tag{2.1}$$

Next, we define the average mutual information of the input and output as a function of the conditional probability density:

$$I(q) = \iint p(\mathbf{x}) q(\mathbf{y}|\mathbf{x}) \log \frac{q(\mathbf{y}|\mathbf{x})}{q(\mathbf{y})} \, d\mathbf{x} \, d\mathbf{y}, \tag{2.2}$$

where $q(\mathbf{y})$ is the marginal distribution found through

$$q(\mathbf{y}) = \int p(\mathbf{x}) q(\mathbf{y}|\mathbf{x}) \, d\mathbf{x}. \tag{2.3}$$

Now, we fix the distortion D. Our goal is to find the corresponding rate $R(D)$. We first find the set of conditional probability densities for which the distortion is less than or equal to D, represented by Q_D:

$$Q_D = \{ q(\mathbf{y}|\mathbf{x}) : d(q) \leq D \}. \tag{2.4}$$

For n successive X_t and Y_t, we find the rate as the minimum mutual information among the X_t and Y_t, limited to those distributions which produce the desired distortion or lower. Note the rate is on a per outcome basis, so the mutual information is divided by n:

$$R_n(D) = \frac{1}{n} \inf_{q \in Q_D} I(q). \tag{2.5}$$

Finally, the rate is the limiting case where the number of outcomes is infinite [40]:

$$R(D) = \lim_{n \to \infty} R_n(D). \tag{2.6}$$

Actually finding the rate, however, is somewhat difficult as it requires finding the infimum of the mutual information over *all* conditional probability densities for which the distortion measure is met. However, if a difference measure is used a simplification of this procedure known as the Shannon Lower Bound may be made. In addition, if it is assumed Gaussian sources are used, and $\rho(\cdot, \cdot)$ is a *squared* difference measure, one may apply the Toeplitz Distribution Theorem to convert the problem from one in the time or spatial domain to one in the Fourier domain. In essence, transform coding is the practical implementation of this step. Finally, applying limiting arguments one may use continuous values in the index of the random process to find (2.7) and (2.8) [40]. In essence, the complicated parameter of conditional probability functions has been replaced with a single parameter, θ.

For the notation that follows lower case letters denote the signals and upper case indicate the Fourier transform. Signals are which are functions of spatial variables x and y are written as $s = s(\lambda)$, where $\lambda = (x, y)$. The resulting Fourier transform is denoted $S = S(\Lambda)$, where $\Lambda = (\omega_x, \omega_y)$. When the third dimension of time is required, the notation is expanded to $s = s(\lambda, t)$, and the Fourier transform as $S = S(\Omega)$, where $\Omega = (\Lambda, \omega_t)$.

2.2.2 Optimum Intraframe Encoding

Below is described an optimum image codec where a mean-square error criterion is used to determine the quality of the output image. Given a two dimensional,

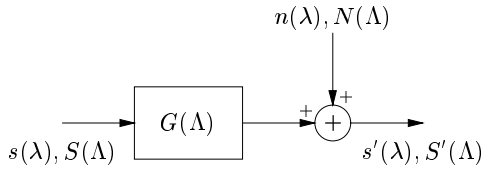


Fig. 2.2. Block diagram of an optimum MSE codec.

stationary, jointly Gaussian, input random process $s = s(\lambda)$, its associated power spectral density $S_{ss}(\Lambda)$ [41], and output of the codec $s' = s'(\lambda)$, for a mean-squared error criterion one can derive the following parameterized representations of the rate-distortion function [40]:

$$D_{\mathrm{O}}^{\theta} = E\left\{(s - s')^2\right\} = \frac{1}{4\pi^2} \iint_{\Lambda} \min\left[\theta, S_{ss}(\Lambda)\right] \, d\Lambda. \tag{2.7}$$

The minimum rate for this distortion is

$$R_{\mathrm{O}}^{\theta} = \frac{1}{8\pi^2} \iint_{\Lambda} \max\left[0, \log_2 \frac{S_{ss}(\Lambda)}{\theta}\right] \, d\Lambda, \tag{2.8}$$

in units of [bits/(unit length)^2]. The variable θ is the generating parameter which takes on all positive real values, and the subscript O indicates the result is optimum with respect to the MSE criterion. In fact, it can be shown these results are equivalent to using the "optimum forward channel" which is shown in Fig. 2.2 [40]. The transfer function $G(\Lambda)$ is represented by

$$G(\Lambda) = \max\left[0, 1 - \frac{\theta}{S_{ss}(\Lambda)}\right] \tag{2.9}$$

and $n(\lambda)$ is an independent, zero mean, Gaussian random process with a power spectral density given by

$$S_{nn}(\Lambda) = \max\left[0, \theta\left(1 - \frac{\theta}{S_{ss}(\Lambda)}\right)\right]. \tag{2.10}$$

In order to facilitate the derivation of the rate-distortion function in Section 2.2.3, two simple but important relations are noted here.

First, the power spectral density of the output of the optimum filter is derived. Given $S(\Lambda)$ and $S'(\Lambda)$ are the Fourier transforms of $s(\lambda)$ and $s'(\lambda)$, respectively, then

$$S'(\Lambda) = G(\Lambda)S(\Lambda) + N(\Lambda). \tag{2.11}$$

Strictly speaking, the Fourier transforms of random processes do not exist. However, for the class of signals we are examining the transforms are a useful shorthand, and we continue the practice here that was also used in [39].

Since $n(\lambda)$ is independent zero mean Gaussian noise,

$$S_{s's'}(\Lambda) = |G(\Lambda)|^2 S_{ss}(\Lambda) + S_{nn}(\Lambda) \tag{2.12}$$

$$= \left|\max\left[0, 1 - \frac{\theta}{S_{ss}(\Lambda)}\right]\right|^2 S_{ss}(\Lambda) + \max\left[0, \theta\left(1 - \frac{\theta}{S_{ss}(\Lambda)}\right)\right] \tag{2.13}$$

$$= \max\left[0, S_{ss}(\Lambda) - \theta\right]. \tag{2.14}$$

We note (2.12) can be derived from (2.11) by transforming back to the time domain and finding the Fourier transform of the correlation of $s'(\lambda)$ with itself.

Next, the power spectral density of the difference of the input and output of the optimum MSE codec is derived. For the system shown in Fig. 2.3,

$$\tilde{S}(\Lambda) = S(\Lambda) - S'(\Lambda) \tag{2.15}$$

$$= \left(1 - G(\Lambda)\right) S(\Lambda) - N(\Lambda) \tag{2.16}$$

and thus

$$S_{\tilde{s}\tilde{s}}(\Lambda) = |1 - G(\Lambda)|^2 S_{ss}(\Lambda) + S_{nn}(\Lambda) \tag{2.17}$$

$$= \left|\min\left[1, \frac{\theta}{S_{ss}(\Lambda)}\right]\right|^2 S_{ss}(\Lambda) + \max\left[0, \theta\left(1 - \frac{\theta}{S_{ss}(\Lambda)}\right)\right] \tag{2.18}$$

$$= \min\left[\theta, S_{ss}(\Lambda)\right]. \tag{2.19}$$

We note this expression is exactly what is required to find the distortion in (2.7).

2.2.3 Alternate Optimum MSE Encoding Models

In this section we explore two alternate encoding models which use *two* of the optimum forward channels shown in Fig. 2.2. These results will allow much easier

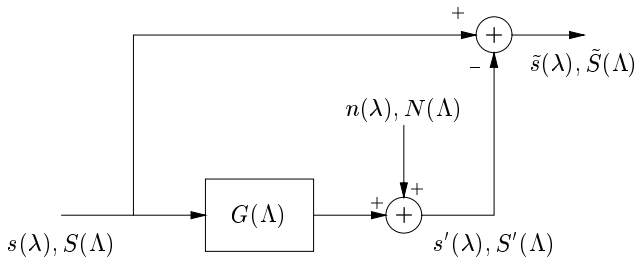


Fig. 2.3. Block diagram of an optimum MSE codec with differential output.

derivation of the rate-distortion function for MCP scalable video compression presented in Section 2.3.

Optimum Layered Encoding

Fig. 2.4 shows the block diagram for a layered codec using two optimum MSE codecs. (The spatial domain notation has been dropped.) The signal is first encoded using an optimum MSE codec, and then the difference between the encoded signal and the original signal is encoded using a second optimum MSE codec.

We first determine the distortion associated with the system, which is designated D_{I} to differentiate it from the distortion found from (2.7). As in (2.7), the distortion is defined to be the expected value of the square difference of the input and output. Using Fig. 2.4 it can be shown the difference between the input and the output is exactly equal to the difference between the input and output of the last codec. Thus, the entire distortion may be measured by simply measuring the distortion at the last stage. Thus

$$D_{\mathrm{I}} \triangleq E\left[(s - s'')^2\right] = E\left[(\tilde{s} - \tilde{s}')^2\right]. \tag{2.20}$$

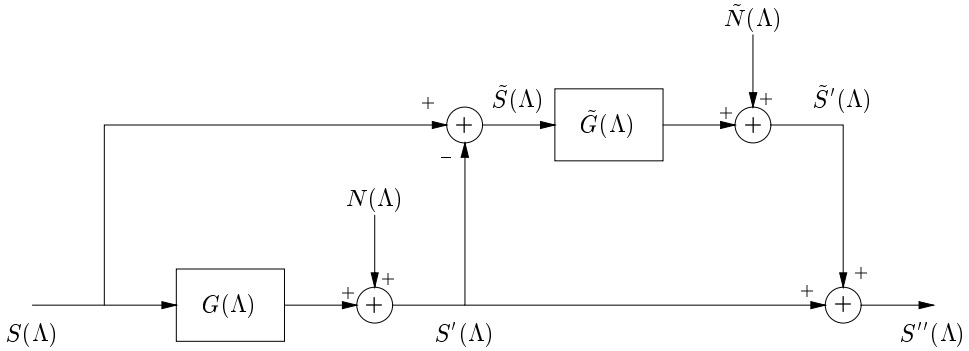


Fig. 2.4. Block diagram an of optimum MSE layered codec.

Consequently,

$$D_{\mathrm{I}}^{\theta,\tilde{\theta}} = \frac{1}{4\pi^2}\iint_{\Lambda} \min\left[\tilde{\theta}, S_{\tilde{s}\tilde{s}}(\Lambda)\right]\, d\Lambda \tag{2.21}$$

$$= \frac{1}{4\pi^2}\iint_{\Lambda} \min\left[\tilde{\theta}, \min\left[\theta, S_{ss}(\Lambda)\right]\right]\, d\Lambda \tag{2.22}$$

and assuming $\tilde{\theta} \leq \theta$,

$$D_{\mathrm{I}}^{\theta,\tilde{\theta}} = \frac{1}{4\pi^2}\iint_{\Lambda} \min\left[\tilde{\theta}, S_{ss}(\Lambda)\right]\, d\Lambda. \tag{2.23}$$

where (2.22) is derived from (2.19) and the variables θ and $\tilde{\theta}$ have been explicitly added to D_{I} to show the dependence of the distortion on these variables. We note if $\tilde{\theta} > \theta$, the system is no longer operating in a layered fashion and (2.23) no longer holds.

Now, the rate of the layered codec, designated R_{I}, must come from two sources: the codec associated with θ, *i.e.*, between s and s', and the codec associated with $\tilde{\theta}$, *i.e.*, between $\tilde{s}$ and $\tilde{s}'$. The total rate is simply the sum of the two individual rates. Consequently, if $\tilde{\theta} \leq \theta$ as in (2.23),

$$R_{\mathrm{I}}^{\theta,\tilde{\theta}} \triangleq \frac{1}{8\pi^2}\iint_{\Lambda} \max\left[0, \log_2 \frac{S_{ss}(\Lambda)}{\theta}\right]\, d\Lambda + \frac{1}{8\pi^2}\iint_{\Lambda} \max\left[0, \log_2 \frac{S_{\tilde{s}\tilde{s}}(\Lambda)}{\tilde{\theta}}\right]\, d\Lambda \tag{2.24}$$

$$= \frac{1}{8\pi^2}\iint_{\Lambda} \left(\max\left[0, \log_2 \frac{S_{ss}(\Lambda)}{\theta}\right] + \max\left[0, \log_2 \frac{\min[\theta, S_{ss}(\Lambda)]}{\tilde{\theta}}\right]\right)\, d\Lambda, \tag{2.25}$$

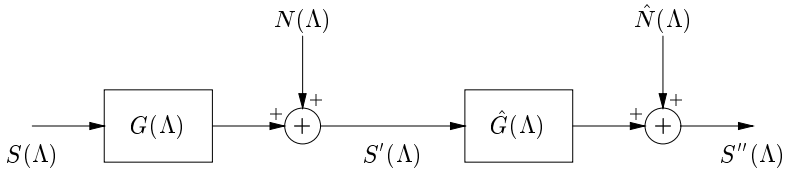


Fig. 2.5. Block diagram of an optimum MSE cascaded codec.

which can be simplified to

$$R_{\mathrm{I}}^{\theta,\tilde{\theta}} = \frac{1}{8\pi^2} \iint_{\Lambda} \max \left[0, \log_2 \frac{S_{ss}(\Lambda)}{\tilde{\theta}} \right] d\Lambda. \tag{2.26}$$

Using (2.23) and (2.26) as compared to (2.7) and (2.8), we can determine for $\tilde{\theta} \leq \theta$ that $D_{\mathrm{I}}^{\theta,\tilde{\theta}} = D_{\mathrm{O}}^{\tilde{\theta}}$ and $R_{\mathrm{I}}^{\theta,\tilde{\theta}} = R_{\mathrm{O}}^{\tilde{\theta}}$. Thus the layered codec has a rate distortion function which is equivalent to a single optimum MSE codec. The usefulness of this derivation will become apparent in Section 2.3.1.

Optimum Cascaded Encoding

The cascaded system shown in Fig. 2.5 is the dual to the one developed in Section 2.2.3. Here we will discover two sources of distortion, but only one place where the rate is determined.

As in (2.7), the distortion of the cascaded system, designated D_{II}, is defined to be the expected value of the square difference of the input and output. Using Fig. 2.5 it can be shown the difference between the input and the output is exactly equal to the following:

$$\begin{aligned} D_{\mathrm{II}} &\triangleq E\left\{ \left(s - s''\right)^2 \right\} && (2.27) \\ &= E\left\{ \left[(s - s') + (s' - s'') \right]^2 \right\} && (2.28) \\ &= E\left\{ (s - s')^2 \right\} + E\left\{ (s' - s'')^2 \right\} + 2E\left\{ (s - s')(s' - s'') \right\} && (2.29) \\ &= E\left\{ (s - s')^2 \right\} + E\left\{ (s' - s'')^2 \right\}, && (2.30) \end{aligned}$$

where (2.30) is only true if $s - s'$ and $s' - s''$ are uncorrelated. While this is in general not true for cascaded systems, we show below this is true when using optimum MSE codecs.

Define $\tilde{s}(\lambda) = s(\lambda) - s'(\lambda)$ and $\hat{s}(\lambda) = s'(\lambda) - s''(\lambda)$. Then $\tilde{s}(\lambda)$ and $\hat{s}(\lambda)$ are uncorrelated if and only if $S_{\tilde{s}\hat{s}}(\Lambda) = 0$ [41]. We know

$$\tilde{S}(\Lambda) = (1 - G(\Lambda))S(\Lambda) - N(\Lambda) \tag{2.31}$$

$$\hat{S}(\Lambda) = (1 - \hat{G}(\Lambda))S'(\Lambda) - \hat{N}(\Lambda) \tag{2.32}$$

$$= (1 - \hat{G}(\Lambda))G(\Lambda)S(\Lambda) + (1 - \hat{G}(\Lambda))N(\Lambda) - \hat{N}(\Lambda). \tag{2.33}$$

Following the same procedures used to derive (2.12), the power spectral density of the cross-correlation of $\tilde{s}(\lambda)$ and $\hat{s}(\lambda)$ can be determined to be

$$S_{\tilde{s}\hat{s}}(\Lambda) = (1 - G(\Lambda)) \left[(1 - \hat{G}(\Lambda))G(\Lambda)\right]^* S_{ss}(\Lambda) - (1 - \hat{G}(\Lambda))^* S_{nn}(\Lambda) \tag{2.34}$$

$$= (1 - \hat{G}^*(\Lambda)) \left[(1 - G(\Lambda))G^*(\Lambda)S_{ss}(\Lambda) - S_{nn}(\Lambda)\right]. \tag{2.35}$$

The expression in square brackets in (2.35) may be evaluated the same way as in Section 2.2.2 and thus

$$[1-G(\Lambda)]G^*(\Lambda)S_{ss}(\Lambda) - S_{nn}(\Lambda) \tag{2.36}$$

$$= \min\left[1, \frac{\theta}{S_{ss}(\Lambda)}\right] \max\left[0, 1 - \frac{\theta}{S_{ss}(\Lambda)}\right] S_{ss}(\Lambda) - \max\left[0, \theta\left(1 - \frac{\theta}{S_{ss}(\Lambda)}\right)\right] \tag{2.37}$$

$$= 0. \tag{2.38}$$

Consequently, $S_{\tilde{s}\hat{s}}(\Lambda) = 0$ for all Λ and thus (2.30) is true.

Using the observation in (2.19), we rewrite (2.30) as

$$D_{\mathrm{II}} = \frac{1}{4\pi^2} \iint_{\Lambda} S_{\tilde{s}\tilde{s}}\, d\Lambda + \frac{1}{4\pi^2} \iint_{\Lambda} S_{\hat{s}\hat{s}}\, d\Lambda \tag{2.39}$$

and thus with (2.7) we find

$$D_{\mathrm{II}}^{\theta,\hat{\theta}} = \frac{1}{4\pi^2} \iint_{\Lambda} \min\left[\theta, S_{ss}(\Lambda)\right] d\Lambda + \frac{1}{4\pi^2} \iint_{\Lambda} \min\left[\hat{\theta}, S_{s's'}(\Lambda)\right] d\Lambda \tag{2.40}$$

$$= \frac{1}{4\pi^2} \iint_{\Lambda} \min\left[\theta, S_{ss}(\Lambda)\right] + \min\left[\hat{\theta}, \max[0, S_{ss}(\Lambda) - \theta]\right] d\Lambda, \tag{2.41}$$

which can be simplified to

$$D_{\mathrm{II}}^{\theta,\hat{\theta}} = \frac{1}{4\pi^2} \iint_{\Lambda} \min\left[\theta + \hat{\theta}, S_{ss}(\Lambda)\right] \, d\Lambda. \tag{2.42}$$

To facilitate the following discussion of the rate of the cascaded system, designated R_{II}, we note (2.8) may be written as

$$R_{\mathrm{O}}^{\theta} = \frac{1}{8\pi^2} \iint_{\Lambda} \log_2 \frac{\max\left[0, S_{ss}(\Lambda) - \theta\right] + \theta}{\theta} \, d\Lambda. \tag{2.43}$$

Since the last stage is the only stage which would be transmitted in a real system, it is tempting to write

$$R_{\mathrm{II}}^{\theta,\hat{\theta}} = \iint_{\Lambda} \log_2 \frac{\max\left[0, S_{s's'}(\Lambda) - \hat{\theta}\right] + \hat{\theta}}{\hat{\theta}} \, d\Lambda. \tag{2.44}$$

The problem with using (2.44) is a magnitude factor of θ which was removed from the power spectral density in the first stage as shown in (2.14). We know in general for all continuous systems the data rate is relative [40]. As shown in (2.43), in (2.8) it is assumed the maximum value of the power spectral density to be transmitted is exactly the maximum value in the input power spectral density, and bits are predicted relative to this value. However, if we use (2.43) directly then the number of bits predicted by (2.44) will be larger than actually required. In effect after the first stage a constant offset is being removed from both the numerator and denominator in the rate computation, and this forces the ratio to be higher at the second stage. To compensate for this, we choose to be relative to the maximum of the input power spectral density, and thus define

$$R_{\mathrm{II}}^{\theta,\hat{\theta}} \triangleq \iint_{\Lambda} \log_2 \frac{\max\left[0, S_{s's'}(\Lambda) - \hat{\theta}\right] + \hat{\theta} + \theta}{\hat{\theta} + \theta} \, d\Lambda. \tag{2.45}$$

where the addition of θ in the numerator and denominator accounts for the magnitude lost after the first stage.

Using the usual substitutions, we find

$$R_{\mathrm{II}}^{\theta,\hat{\theta}} = \frac{1}{8\pi^2} \iint_{\Lambda} \log_2 \frac{\max\left[0, S_{ss}(\Lambda) - \hat{\theta} - \theta\right] + \hat{\theta} + \theta}{\hat{\theta} + \theta} \, d\Lambda \tag{2.46}$$

$$= \frac{1}{8\pi^2} \iint_{\Lambda} \max\left[0, \log_2 \frac{S_{ss}(\Lambda)}{\hat{\theta} + \theta}\right] \, d\Lambda. \tag{2.47}$$

Using (2.42) and (2.47) as compared to (2.7) and (2.8), we can determine $D_{\mathrm{II}}^{\theta,\hat{\theta}} = D_{\mathrm{O}}^{\theta+\hat{\theta}}$ and $R_{\mathrm{II}}^{\theta,\hat{\theta}} = R_{\mathrm{O}}^{\theta+\hat{\theta}}$. Thus the cascaded system is also has a rate-distortion function which is equivalent to a single optimum MSE codec. The usefulness of this derivation will become apparent in Section 2.3.2.

2.2.4 Interframe Encoding

This section is essentially a summation of [39] which describes the properties of a MCP non-scalable video system using an optimum MSE codec and displacement estimates. The system variables are now extended to include time, *e.g.*, $s = s(\lambda, t)$, and the corresponding Fourier transform is designated by $S = S(\Lambda, \omega_t) = S(\Omega)$.

The non-scalable MCP system which was described in Section 2.1 was modeled in [39] as shown in Fig. 2.6. The codec is the optimum MSE codec introduced in Section 2.2.2. The MCP loop is represented by the feedback path below the optimum MSE codec. Note the motion vectors are not represented on this diagram; comments on this omission appear later in this section. Essentially, the properties of the MCP loop are captured by $H(\Omega)$. This stochastic filter is a combination delay, motion compensation, and spatial filter, and fulfills in a mathematically tractable way the modeling of the MCP loop. Similar to Fig. 2.1, on the right-hand side of the diagram the MCP loop is reversed and produces the proper signal on the output. The exact representation of $H(\Omega)$ is described later in this section.

Since $s - s' = e - e'$ (or equivalently $S(\Omega) - S'(\Omega) = E(\Omega) - E'(\Omega)$), the rate-distortion function of the interframe codec can be obtained by substituting $S_{ee}(\Lambda)$ for $S_{ss}(\Lambda)$ in (2.7) and (2.8) [39]. Thus we may state for the non-scalable MCP video codec

$$D_{\mathrm{O}}^{\theta} = E\left\{(e - e')^2\right\} = \frac{1}{4\pi^2} \iint_{\Lambda} \min\left[\theta, S_{ee}^{\theta}(\Lambda)\right] \, d\Lambda \tag{2.48}$$

$$R_{\mathrm{O}}^{\theta} = \frac{1}{8\pi^2} \iint_{\Lambda} \max\left[0, \log_2 \frac{S_{ee}^{\theta}(\Lambda)}{\theta}\right] \, d\Lambda, \tag{2.49}$$

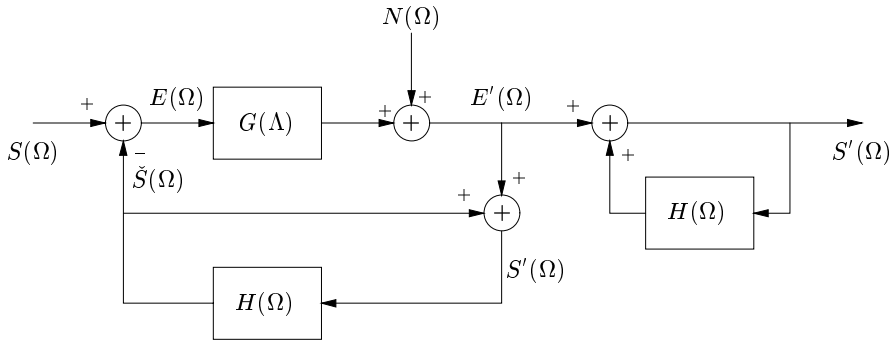


Fig. 2.6. Block diagram of an MCP optimum MSE codec.

where the dependence of $S_{ee}(\Lambda)$ on θ is explicitly denoted. This dependence is explored more fully below.

Unlike $S_{ss}(\Lambda)$, $S^{\theta}_{ee}(\Lambda)$ is not given and must be determined based on the MCP method. In fact, since

$$\begin{aligned} E(\Omega) =& \frac{1-H(\Omega)}{1-H(\Omega)+H(\Omega)G(\Omega)}S(\Omega) \\ & - \frac{H(\Omega)}{1-H(\Omega)+H(\Omega)G(\Omega)}N(\Omega), \end{aligned} \tag{2.50}$$

it can be determined given

$$S^{\theta}_{ee}(\Lambda) = \frac{\Delta_t}{2\pi}\int_0^{\frac{2\pi}{\Delta_t}} S^{\theta}_{ee}(\Omega)\, d\omega_t, \tag{2.51}$$

then [39]

$$\begin{aligned} S^{\theta}_{ee}(\Lambda) =& \frac{\Delta_t}{2\pi}\int_0^{\frac{2\pi}{\Delta_t}} E\left\{\left|\frac{1-H(\Omega)}{1-H(\Omega)\min\left[1,\dfrac{\theta}{S^{\theta}_{ee}(\Lambda)}\right]}\right|^2\right\} S_{ss}(\Omega)\, d\omega_t \\ & + \frac{\Delta_t}{2\pi}\max\left[0, 1-\frac{\theta}{S^{\theta}_{ee}(\Lambda)}\right]\int_0^{\frac{2\pi}{\Delta_t}} E\left\{\left|\frac{H(\Omega)}{1-H(\Omega)\min\left[1,\dfrac{\theta}{S^{\theta}_{ee}(\Lambda)}\right]}\right|^2\right\} d\omega_t. \end{aligned} \tag{2.52}$$

Since (2.52) is at best difficult to solve analytically, the following observations have been made [39]. If $S_{ee}^{\theta}(\Lambda) \gg \theta$ (designated Case I)

$$\begin{aligned} S_{ee}^{\mathrm{I},\theta}(\Lambda) =& \frac{\Delta_t}{2\pi} \int_0^{\frac{2\pi}{\Delta_t}} E\left\{|1 - H(\Omega)|^2\right\} S_{ss}(\Omega)\, d\omega_t \\ &+ \frac{\Delta_t}{2\pi} \int_0^{\frac{2\pi}{\Delta_t}} E\left\{|H(\Omega)|^2\right\} d\omega_t. \end{aligned} \tag{2.53}$$

It is also clear if $S_{ss}(\Lambda) \leq \theta$ (designated Case II)

$$S_{ee}^{\mathrm{II},\theta}(\Lambda) = S_{ss}(\Lambda). \tag{2.54}$$

If the transition between Case I and Case II covers only a small spatial frequency range, a reasonable approximation to $S_{ee}^{\theta}(\Lambda)$ would be [39]

$$S_{ee}^{\theta}(\Lambda) \approx S_{ee}^{\mathrm{appr},\theta}(\Lambda) \triangleq \begin{cases} \max\left[\theta, S_{ee}^{\mathrm{I},\theta}(\Lambda)\right] & \{\Lambda : S_{ss}(\Lambda) > \theta\} \\ S_{ss}(\Lambda) & \{\Lambda : S_{ss}(\Lambda) \leq \theta\}. \end{cases} \tag{2.55}$$

The filter $H(\Omega)$ is the Fourier transform of the time delay and motion compensation in the MCP loop, along with a spatial filter whose utility will be seen shortly. In [39] it is represented as

$$H(\Omega) = H(\Lambda, \omega_t) = F(\Lambda) \exp\left(-j\Lambda \cdot \hat{d} - j\omega_t \Delta_t\right), \tag{2.56}$$

where $\hat{d}$ is the two-dimensional estimated displacement vector and $t - \Delta_t$ is the time interval from which the estimated displacement vector is computed. For constant, translatory displacement $S_{ee}^{\mathrm{I},\theta}(\Lambda)$ is found to be

$$S_{ee}^{\mathrm{I},\theta}(\Lambda) = S_{ss}(\Lambda)\left[1 - 2\Re\left\{F(\Lambda)P(\Lambda)\right\} + |F(\Lambda)|^2\right] + \theta|F(\Lambda)|^2, \tag{2.57}$$

where $P(\Lambda)$ is the 2-D Fourier transform of the probability density function $p_{\Delta d}(\Delta d)$ with $\Delta d = d - \hat{d}$, and d is the known displacement. There are three cases of interest based on the function $F(\Lambda)$.

Intraframe encoding

This occurs when no motion compensation is used, and corresponds to

$$F(\Lambda) = 0. \tag{2.58}$$

Clearly for this case,

$$S_{ee}^{\mathrm{I},\theta}(\Lambda) = S_{ss}(\Lambda). \tag{2.59}$$

Motion Compensation with No Spatial Prediction Filter

This case corresponds most closely to DPCM—no attempt is made to adjust the motion compensation spatially, and thus

$$F(\Lambda) = 1. \tag{2.60}$$

Consequently,

$$S_{ee}^{\mathrm{I},\theta}(\Lambda) = 2S_{ss}(\Lambda)\left[1 - \Re\{P(\Lambda)\}\right] + \theta. \tag{2.61}$$

Optimum Spatial Filtering

Here, $S_{ee}^{\mathrm{I},\theta}(\Lambda)$ is optimized to be a minimum for each spatial frequency. It was found in [39] that this is true when

$$F(\Lambda) = P^*(\Lambda)\frac{S_{ss}(\Lambda)}{S_{ss}(\Lambda) + \theta} \tag{2.62}$$

and thus

$$S_{ee}^{\mathrm{I},\theta}(\Lambda) = S_{ss}(\Lambda)\left[1 - \frac{|P(\Lambda)|^2 S_{ss}(\Lambda)}{S_{ss}(\Lambda) + \theta}\right]. \tag{2.63}$$

In this analysis the data rate needed to represent the motion vectors is ignored, since to a first approximation the entropy of the encoded motion vectors is approximately the same for all data rates and is low in comparison to the total data rate [42]. In effect, the motion vectors just add a constant, positive offset to the data rate. For very low data rate encoding and multi-loop MCP (*e.g.,* [31]) this approximation begins to break down and a model of the entropy of the motion vectors would produce more accurate results in these cases [42].

2.3 Rate-distortion function for MCP Scalable Video Compression

In Section 2.2.4, a rate-distortion model for MCP video compression was described. In this section we take this description and develop two rate-distortion functions, using as guides the theory developed in Sections 2.2.3 and 2.2.3. The first rate-distortion function, designated Case I, is a model for those techniques for which there is no prediction drift, *i.e.,* when operating above the MCP rate. The second rate distortion function, designated Case II, models prediction drift, which occurs when operating below the MCP rate.

2.3.1 Case I: Scalable Video Operating above the MCP Rate

When decoding scalable video above the MCP rate, there are in essence two data sources: a MCP base layer, and an enhancement layer which is an encoding of the difference between the original signal and the base layer signal without MCP, *e.g.,* MPEG-4 FGS [10]. In *SAMCoW* the two signals are implicitly combined through the use of a single codec and a marker which designates the end of the base layer bit stream [17, 43]. In either case there is no prediction drift because the enhancement layer does not depend on previous frames. In order to use the layered codec model described in Section 2.2.3, we note, similar to the analysis in Section 2.2.4, for a MCP system $s - s' = e - e'$. Consequently, we can model this system as shown in Fig. 2.7 where the lower section is the standard MCP codec shown in Fig. 2.6 and the upper section, where the variables have tilde accents, encodes the difference between the input signal and the output of the standard MCP codec.

Using (2.20) and (2.23), it can be shown the only modification needed is to substitute $S_{ee}^{\theta}(\Lambda)$ for $S_{ss}(\Lambda)$. Consequently, for $\tilde{\theta} \leq \theta$,

$$D_{\mathrm{I}}^{\theta,\tilde{\theta}} = \frac{1}{4\pi^2} \iint_{\Lambda} \min\left[\tilde{\theta}, S_{ee}^{\theta}(\Lambda)\right] d\Lambda \tag{2.64}$$

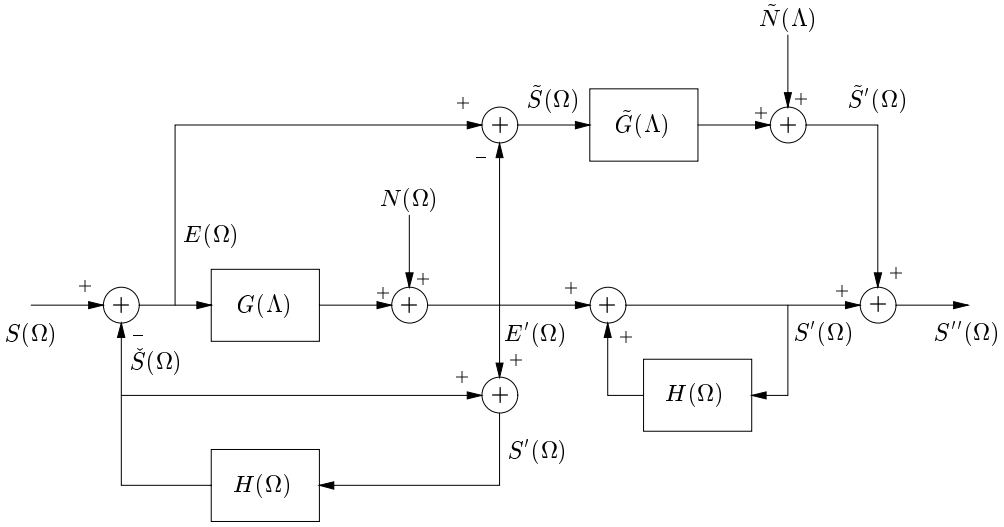


Fig. 2.7. Block diagram of an MCP scalable video codec operating above the MCP rate.

and

$$R_{\mathrm{I}}^{\theta,\tilde{\theta}} = \frac{1}{8\pi^2} \iint_{\Lambda} \max \left[0, \log_2 \frac{S_{ee}^{\theta}(\Lambda)}{\tilde{\theta}} \right] d\Lambda. \tag{2.65}$$

It is instructive to compare Case I with the non-scalable MCP system defined by (2.48) and (2.49). We know from Section 2.2.3 that without MCP the optimum layered system is exactly equivalent to a single codec system which uses $\tilde{\theta}$ as the parameter. Substituting $\tilde{\theta}$ for θ in (2.48) and (2.49) finds the optimum distortion and rate assuming full knowledge of the motion vectors for each $\tilde{\theta}$. As described in Section 2.1, while this is not possible in practice, it represents the optimum point on the rate-distortion curve for both scalable and non-scalable systems. In order to quantify this difference between the scalable codec and the ideal (albeit unobtainable) scalable codec, we define two functions: $\Delta D_{\mathrm{I}}^{\theta,\tilde{\theta}}$ and $\Delta R_{\mathrm{I}}^{\theta,\tilde{\theta}}$. The function $\Delta D_{\mathrm{I}}^{\theta,\tilde{\theta}}$ is the difference of the distortion of a scalable video codec operating above the MCP rate and the ideal scalable video codec, which is represented by the non-scalable codec

operating with the same $\tilde{\theta}$. The function $\Delta R_{\mathrm{I}}^{\theta,\tilde{\theta}}$ is defined similarly, except the rate is used instead of the distortion. Consequently, we define the following when $\tilde{\theta} \leq \theta$:

$$\Delta D_{\mathrm{I}}^{\theta,\tilde{\theta}} \triangleq D_{\mathrm{I}}^{\theta,\tilde{\theta}} - D_{\mathrm{O}}^{\tilde{\theta}} \tag{2.66}$$

$$= \frac{1}{4\pi^2} \iint_{\Lambda} \min\left[\tilde{\theta}, S_{ee}^{\theta}(\Lambda)\right] - \min\left[\tilde{\theta}, S_{ee}^{\tilde{\theta}}(\Lambda)\right] d\Lambda \tag{2.67}$$

and

$$\Delta R_{\mathrm{I}}^{\theta,\tilde{\theta}} \triangleq R_{\mathrm{I}}^{\theta,\tilde{\theta}} - R_{\mathrm{O}}^{\tilde{\theta}} \tag{2.68}$$

$$= \frac{1}{8\pi^2} \iint_{\Lambda} \max\left[0, \log_2 \frac{S_{ee}^{\theta}(\Lambda)}{\tilde{\theta}}\right] - \max\left[0, \log_2 \frac{S_{ee}^{\tilde{\theta}}(\Lambda)}{\tilde{\theta}}\right] d\Lambda. \tag{2.69}$$

These results may be further analyzed dependent on the frequency of interest:

$$\Delta D_{\mathrm{I}}^{\theta,\tilde{\theta}} = \begin{cases} 0 & \left\{\Lambda : S_{ee}^{\theta}(\Lambda) > \tilde{\theta} \text{ and } S_{ee}^{\tilde{\theta}}(\Lambda) > \tilde{\theta}\right\} \\ \dfrac{1}{4\pi^2} \displaystyle\iint_{\Lambda} \left(\tilde{\theta} - S_{ss}(\Lambda)\right) d\Lambda & \left\{\Lambda : S_{ee}^{\theta}(\Lambda) > \tilde{\theta} \text{ and } S_{ee}^{\tilde{\theta}}(\Lambda) \leq \tilde{\theta}\right\} \\ \dfrac{1}{4\pi^2} \displaystyle\iint_{\Lambda} \left(S_{ss}(\Lambda) - \tilde{\theta}\right) d\Lambda & \left\{\Lambda : S_{ee}^{\theta}(\Lambda) \leq \tilde{\theta} \text{ and } S_{ee}^{\tilde{\theta}}(\Lambda) > \tilde{\theta}\right\} \\ 0 & \left\{\Lambda : S_{ee}^{\theta}(\Lambda) \leq \tilde{\theta} \text{ and } S_{ee}^{\tilde{\theta}}(\Lambda) \leq \tilde{\theta}\right\} \end{cases}, \tag{2.70}$$

and similarly,

$$\Delta R_{\mathrm{I}}^{\theta,\tilde{\theta}} = \begin{cases} \dfrac{1}{8\pi^2} \displaystyle\iint_{\Lambda} \log_2 \frac{S_{ee}^{\theta}(\Lambda)}{S_{ee}^{\tilde{\theta}}(\Lambda)} d\Lambda & \left\{\Lambda : S_{ee}^{\theta}(\Lambda) > \tilde{\theta} \text{ and } S_{ee}^{\tilde{\theta}}(\Lambda) > \tilde{\theta}\right\} \\ \dfrac{1}{8\pi^2} \displaystyle\iint_{\Lambda} \log_2 \frac{S_{ee}^{\theta}(\Lambda)}{\tilde{\theta}} d\Lambda & \left\{\Lambda : S_{ee}^{\theta}(\Lambda) > \tilde{\theta} \text{ and } S_{ee}^{\tilde{\theta}}(\Lambda) \leq \tilde{\theta}\right\} \\ -\dfrac{1}{8\pi^2} \displaystyle\iint_{\Lambda} \log_2 \frac{S_{ee}^{\tilde{\theta}}(\Lambda)}{\tilde{\theta}} d\Lambda & \left\{\Lambda : S_{ee}^{\theta}(\Lambda) \leq \tilde{\theta} \text{ and } S_{ee}^{\tilde{\theta}}(\Lambda) > \tilde{\theta}\right\} \\ 0 & \left\{\Lambda : S_{ee}^{\theta}(\Lambda) \leq \tilde{\theta} \text{ and } S_{ee}^{\tilde{\theta}}(\Lambda) \leq \tilde{\theta}\right\} \end{cases}, \tag{2.71}$$

where we have used (2.54) to eliminate $S_{ee}^{\tilde{\theta}}(\Lambda)$ from (2.70).

In the first case of (2.70), if after motion compensation the power spectral density is greater than $\tilde{\theta}$, the change in distortion between a non-scalable optimum system and Case I is exactly zero. The corresponding change in rate in (2.71) is a measure of how much more effective the motion compensation prediction is given the extra information afforded by the fact that $\tilde{\theta} \leq \theta$. In general, we would predict $S_{ee}^{\tilde{\theta}}(\Lambda) \leq S_{ee}^{\theta}(\Lambda)$

since more information is available to the encoder to perform motion-compensated prediction. Consequently, we may eliminate Case 3 in both (2.70) and (2.71). Assuming a properly performing motion-compensating predictor, then $\Delta R_{\mathrm{I}}^{\theta,\tilde{\theta}} \geq 0$, and thus a scalable video codec will always perform worse than or equal to an equivalent non-scalable system. These issues are more fully examined in Section 2.4.

2.3.2 Case II: Scalable Video Operating below the MCP Rate

When scalable video is used below the MCP rate the enhancement layer is completely eliminated and only part of the base layer information is transmitted. For example, in *SAMCoW* this is accomplished in practice by simply truncating the base layer bit stream at the point at which a sufficient number bits to match the desired data rate have been sent. The modeling of this method of decoding is the dual of that shown in Section 2.3.1. Here there is only one data source, but now there are two sources of distortion: one from the usual source of the encoder in the MCP loop, and another because the entire bit stream is not being sent. Because of the latter, there is a mismatch of reference frames in the encoder and decoder, and prediction drift occurs. We model this phenomenon as two independent optimum MSE codecs in cascade. The second optimal MSE codec simply performs the truncation in an optimal way. The block diagram for this codec is shown in Fig. 2.8. The usual encoding and decoding loops of Fig. 2.6 are shown with a second optimal MSE codec inserted between them. Thus, the problem is reduced to determining the effect of the second codec on the performance of the MCP loop.

To determine the distortion, designated D_{II}, we note (2.27) through (2.29) are still true. It remains, then to find if the random processes $s(\lambda,t) - s'(\lambda,t)$ and $s'(\lambda,t) - s''(\lambda,t)$ are uncorrelated for (2.30) to be true. Proceeding as in Section 2.2.3, define $\tilde{s}(\lambda,t) = s(\lambda,t) - s'(\lambda,t)$ and $\hat{s}(\lambda,t) = s'(\lambda,t) - s''(\lambda,t)$. Then, using Fig. 2.8,

$$\tilde{S}(\Omega) = E(\Omega) - E'(\Omega) \tag{2.72}$$

$$= (1 - G(\Omega))E(\Omega) - N(\Omega). \tag{2.73}$$

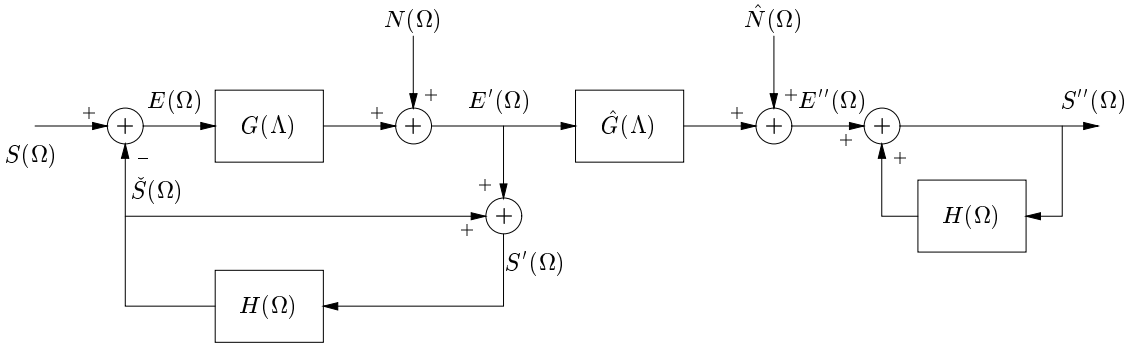


Fig. 2.8. Block diagram of an MCP scalable video codec operating below the MCP rate.

Finding $\hat{S}(\Omega)$ proceeds slightly differently than in Section 2.2.3. First, we note the motion compensation loop on the decoder may be modeled as a filter with a system response of $1/(1-H(\Omega))$. Since this is a linear time-invariant system, we can move this block ahead of the optimum MSE codec as shown in Fig. 2.9.

Thus,

$$S''(\Omega) = \hat{G}(\Omega)S'(\Omega) + \frac{1}{1-H(\Omega)}\hat{N}(\Omega), \tag{2.74}$$

and consequently,

$$\hat{S}(\Omega) = S'(\Omega) - S''(\Omega) \tag{2.75}$$

$$= (1-\hat{G}(\Omega))S'(\Omega) - \frac{1}{1-H(\Omega)}\hat{N}(\Omega) \tag{2.76}$$

$$= (1-\hat{G}(\Omega))\frac{1}{1-H(\Omega)}E'(\Omega) - \frac{1}{1-H(\Omega)}\hat{N}(\Omega). \tag{2.77}$$

Following the same steps for deriving the cross-correlation power spectral density and using the fact that the MCP estimates are independent,

$$S_{\tilde{s}\hat{s}}(\Omega) = (1-G(\Omega))\left[(1-\hat{G}(\Omega))G(\Omega)E\left\{\frac{1}{1-H(\Omega)}\right\}\right]^* S_{ee}(\Omega) \tag{2.78}$$

$$- \left[(1-\hat{G}(\Omega))E\left\{\frac{1}{1-H(\Omega)}\right\}\right]^* S_{nn}(\Omega)$$

$$= (1-\hat{G}^*(\Lambda))E\left\{\frac{1}{1-H(\Omega)}\right\}^* \left[(1-G(\Lambda))G^*(\Lambda)S_{ee}(\Lambda) - S_{nn}(\Lambda)\right], \tag{2.79}$$

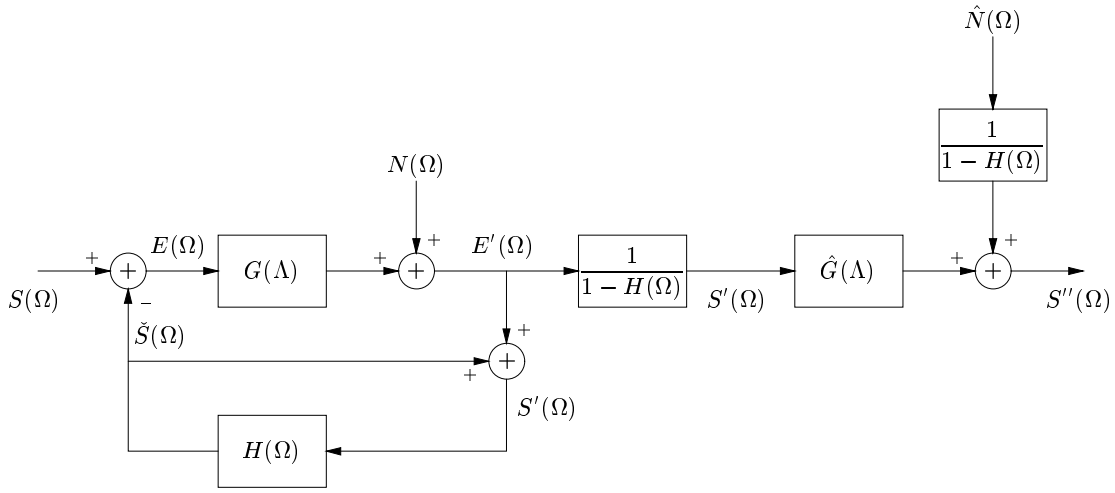


Fig. 2.9. Block diagram of of an MCP scalable video codec operating below the MCP rate with an equivalent signal processing block order.

where $G(\Omega) = G(\Lambda)$, $S_{nn}(\Omega) = S_{nn}(\Lambda)$, and $S_{ee}(\Omega) = S^{\theta}_{ee}(\Lambda)$ as described in Section 2.2.4. Since the expression in the square brackets has exactly the same form as (2.37) with $S^{\theta}_{ee}(\Lambda)$ substituted for $S_{ss}(\Lambda)$, $S_{\tilde{s}\tilde{s}}(\Omega) = 0$, and thus $\tilde{s}(\lambda, t)$ and $\hat{s}(\lambda, t)$ are uncorrelated. Consequently, (2.30) is true, as is (2.39). Thus to find D_{II} the only task remains is to find $S_{\hat{s}\hat{s}}(\Omega)$, since $S_{\tilde{s}\tilde{s}}(\Omega)$ is already known through (2.48).

Now, $S_{\hat{s}\hat{s}}(\Omega)$ can be derived from (2.77):

$$S_{\hat{s}\hat{s}}(\Omega) = E\left\{\left|\frac{1}{1-H(\Omega)}\right|^2\right\}|1-\hat{G}(\Omega)|^2 S_{e'e'}(\Omega) + E\left\{\left|\frac{1}{1-H(\Omega)}\right|^2\right\} S_{\hat{n}\hat{n}}(\Omega). \quad (2.80)$$

Since

$$S_{e'e'}(\Omega) = S_{e'e'}(\Lambda) = \max\left[0, S^{\theta}_{ee}(\Lambda) - \theta\right] \quad (2.81)$$

$$\hat{G}(\Omega) = \hat{G}(\Lambda) = \max\left[0, 1 - \frac{\hat{\theta}}{S_{e'e'}(\Lambda)}\right] \quad (2.82)$$

$$= \max\left[0, 1 - \frac{\hat{\theta}}{\max\left[0, S^{\theta}_{ee}(\Lambda) - \theta\right]}\right]$$

$$S_{\hat{n}\hat{n}}(\Lambda) = \max\left[0, \hat{\theta}\left(1 - \frac{\hat{\theta}}{\max\left[0, S^{\theta}_{ee}(\Lambda) - \theta\right]}\right)\right], \quad (2.83)$$

then

$$S_{\hat{s}\hat{s}}(\Omega) = E\left\{\left|\frac{1}{1-H(\Omega)}\right|^2\right\}\left(\left|\min\left[1, \frac{\hat{\theta}}{\max\left[0, S_{ee}^{\theta}(\Lambda)-\theta\right]}\right]\right|^2 \max\left[0, S_{ee}^{\theta}(\Lambda)-\theta\right] + \max\left[0, \hat{\theta}\left(1-\frac{\hat{\theta}}{\max\left[0, S_{ee}^{\theta}(\Lambda)-\theta\right]}\right)\right]\right). \tag{2.84}$$

Now, since the expression on the right hand side of (2.84), excluding the expected value, is in the form of (2.18),

$$S_{\hat{s}\hat{s}}(\Omega) = E\left\{\left|\frac{1}{1-H(\Omega)}\right|^2\right\}\min\left[\hat{\theta}, \max\left[0, S_{ee}^{\theta}(\Lambda)-\theta\right]\right]. \tag{2.85}$$

Since

$$S_{\hat{s}\hat{s}}(\Lambda) = \frac{\Delta_t}{2\pi}\int_0^{\frac{2\pi}{\Delta_t}} S_{\hat{s}\hat{s}}(\Omega)\, d\omega_t, \tag{2.86}$$

we find

$$= \frac{\Delta_t}{2\pi}\int_0^{\frac{2\pi}{\Delta_t}} E\left\{\left|\frac{1}{1-H(\Omega)}\right|^2\right\} d\omega_t \min\left[\hat{\theta}, \max\left[0, S_{ee}^{\theta}(\Lambda)-\theta\right]\right], \tag{2.87}$$

where we have used the assumption for (2.87) that the output of the first optimal MSE codec at $E'(\Omega)$ is stationary with respect to time and is not a function of ω_t.

The integral on the right hand side of (2.87) can be examined for various spatial filters. From (2.56) we know if $F(\Lambda) = 0$, then the integral multiplied by $\frac{\Delta_t}{2\pi}$ is unity. For the case of $F(\Lambda) = 1$, however, we must examine the integrand. Note

$$\frac{1}{1-\exp\left(-j\Lambda\cdot\hat{d} - j\omega_t\Delta_t\right)} \tag{2.88}$$

is undefined when

$$\Lambda\cdot\hat{d} + \omega_t\Delta_t = k2\pi, \tag{2.89}$$

where $k = \ldots, -2, -1, 0, 1, 2, \ldots$. As a consequence, the method outlined here cannot be used to analyze the rate-distortion function with $F(\Lambda) = 1$. Practically speaking,

the errors induced by not using some form of spatial filter almost eliminates being able to use the bit stream below the rate at which MCP was performed.

However, if we use the optimum prediction filter as given in (2.63) and note $|P(\Lambda)| \leq 1$ (see for example [41]), then

$$|F(\Lambda)| = \left| \frac{P(\Lambda)S_{ss}(\Lambda)}{S_{ss}(\Lambda) + \theta} \right| < 1 \tag{2.90}$$

and the integral is defined. Consequently, the use of an optimum prediction filter is a sufficient condition to guarantee stability of the system irrespective of the input power spectral density and displacement probability density function.

In fact, if $|F(\Lambda)| < 1$, the integral can be solved in closed-form. We note first the integrand may be written as the following:

$$\frac{1}{|1 - H(\Omega)|^2} = \frac{1}{1 + |H(\Omega)|^2 - 2\Re\{H(\Omega)\}} \tag{2.91}$$

$$= \frac{1}{1 + |F(\Lambda)|^2 - 2|F(\Lambda)|\cos(\omega_t \Delta_t + \phi)}, \tag{2.92}$$

where $\phi = -\arg F + \Lambda \cdot \hat{d}$. Using standard integral tables we find the following fact [44]:

$$\frac{1}{2\pi} \int_0^{2\pi} \frac{1}{1 + a^2 - 2a\cos(x + \alpha)}\, dx = \frac{1}{1 - a^2},\ 0 \leq a < 1. \tag{2.93}$$

Thus, if we switch the order of integration of ω_t and the expectation, we find

$$E\left\{ \frac{\Delta_t}{2\pi} \int_0^{\frac{2\pi}{\Delta_t}} \left| \frac{1}{1 - H(\Omega)} \right|^2 d\omega_t \right\} = \frac{1}{1 - |F(\Lambda)|^2}. \tag{2.94}$$

Consequently,

$$S_{\hat{s}\hat{s}}(\Lambda) = \frac{1}{1 - |F(\Lambda)|^2} \min\left[\hat{\theta}, \max\left[0, S_{ee}^{\theta}(\Lambda) - \theta\right] \right]. \tag{2.95}$$

Substituting (2.95) into (2.39) we find

$$D_{\mathrm{II}}^{\theta,\tilde{\theta}-\theta} = \frac{1}{4\pi^2} \iint_{\Lambda} \min\left[\theta, S_{ee}^{\theta}(\Lambda)\right] + \frac{1}{1 - |F(\Lambda)|^2} \min\left[\tilde{\theta} - \theta, \max\left[0, S_{ee}^{\theta}(\Lambda) - \theta\right] \right] d\Lambda, \tag{2.96}$$

where $\tilde{\theta} = \theta + \hat{\theta}$ has been substituted into (2.96) for easier comparison to Case I.

Determining the rate, designated R_{II}, is straightforward and directly follows the steps in Section 2.2.3, where $S_{ee}^{\theta}(\Lambda)$ is substituted for $S_{ss}(\Lambda)$:

$$R_{\mathrm{II}}^{\theta,\tilde{\theta}-\theta} = \frac{1}{8\pi^2}\iint_{\Lambda} \log_2 \frac{\max[0, S_{ee}^{\theta}(\Lambda) - \tilde{\theta}] + \tilde{\theta}}{\tilde{\theta}}\, d\Lambda \tag{2.97}$$

$$= \frac{1}{8\pi^2}\iint_{\Lambda} \max\left[0, \log_2 \frac{S_{ee}^{\theta}(\Lambda)}{\tilde{\theta}}\right]\, d\Lambda. \tag{2.98}$$

Similar to Section 2.3.1, we may determine the difference in distortion and rate with respect to a non-scalable system. Thus, for $\tilde{\theta} > \theta$,

$$\Delta D_{\mathrm{II}}^{\theta,\tilde{\theta}} \triangleq D_{\mathrm{II}}^{\theta,\tilde{\theta}-\theta} - D_{\mathrm{O}}^{\tilde{\theta}} \tag{2.99}$$

$$= \frac{1}{4\pi^2}\iint_{\Lambda} \min\left[\theta, S_{ee}^{\theta}(\Lambda)\right] + \frac{1}{1-|F(\Lambda)|^2}\min\left[\tilde{\theta}-\theta, \max\left[0, S_{ee}^{\theta}(\Lambda)-\theta\right]\right] - \min\left[\tilde{\theta}, S_{ee}^{\tilde{\theta}}(\Lambda)\right]\, d\Lambda \tag{2.100}$$

and

$$\Delta R_{\mathrm{II}}^{\theta,\tilde{\theta}} \triangleq R_{\mathrm{II}}^{\theta,\tilde{\theta}-\theta} - R_{\mathrm{O}}^{\tilde{\theta}} \tag{2.101}$$

$$= \frac{1}{8\pi^2}\iint_{\Lambda} \max\left[0, \log_2 \frac{S_{ee}^{\theta}(\Lambda)}{\tilde{\theta}}\right] - \max\left[0, \log_2 \frac{S_{ee}^{\tilde{\theta}}(\Lambda)}{\tilde{\theta}}\right]\, d\Lambda. \tag{2.102}$$

Again, similar to Section 2.3.1, we can further analyze these results based on the frequency of interest.

For the $\Delta D_{\text{II}}^{\theta,\tilde{\theta}}$ cases, there is one extra case which we need to examine as compared to (2.71), namely whether S_{ee}^{θ} is greater than or less than θ. Previously, there was no case dependence on θ, just on S_{ee}^{θ}. Thus we can determine

$$\Delta D_{\text{II}}^{\theta,\tilde{\theta}} = \begin{cases} \dfrac{1}{4\pi^2}\displaystyle\iint_{\Lambda} \dfrac{|F(\Lambda)|^2}{1-|F(\Lambda)|^2}\left(\tilde{\theta}-\theta\right)\, d\Lambda & \left\{\Lambda : S_{ee}^{\theta}(\Lambda) > \tilde{\theta} \text{ and } S_{ee}^{\tilde{\theta}}(\Lambda) > \tilde{\theta}\right\} \\ \dfrac{1}{4\pi^2}\displaystyle\iint_{\Lambda} \dfrac{|F(\Lambda)|^2}{1-|F(\Lambda)|^2}\left(\tilde{\theta}-\theta\right) & \left\{\Lambda : S_{ee}^{\theta}(\Lambda) > \tilde{\theta} \text{ and } S_{ee}^{\tilde{\theta}}(\Lambda) \le \tilde{\theta}\right\} \\ \qquad + \tilde{\theta} - S_{ss}(\Lambda)\, d\Lambda & \\ \dfrac{1}{4\pi^2}\displaystyle\iint_{\Lambda} \dfrac{|F(\Lambda)|^2}{1-|F(\Lambda)|^2}\left(S_{ee}^{\theta}(\Lambda)-\theta\right) & \left\{\Lambda : \theta < S_{ee}^{\theta}(\Lambda) \le \tilde{\theta} \text{ and } S_{ee}^{\tilde{\theta}}(\Lambda) > \tilde{\theta}\right\} \\ \qquad + S_{ee}^{\theta}(\Lambda) - \tilde{\theta}\, d\Lambda & \\ \dfrac{1}{4\pi^2}\displaystyle\iint_{\Lambda} S_{ss}(\Lambda) - \tilde{\theta}\, d\Lambda & \left\{\Lambda : S_{ee}^{\theta}(\Lambda) \le \theta \text{ and } S_{ee}^{\tilde{\theta}}(\Lambda) > \tilde{\theta}\right\} \\ \dfrac{1}{4\pi^2}\displaystyle\iint_{\Lambda} \dfrac{|F(\Lambda)|^2}{1-|F(\Lambda)|^2}\left(S_{ee}^{\theta}(\Lambda)-\theta\right) & \left\{\Lambda : \theta < S_{ee}^{\theta}(\Lambda) \le \tilde{\theta} \text{ and } S_{ee}^{\tilde{\theta}}(\Lambda) \le \tilde{\theta}\right\} \\ \qquad + S_{ee}^{\theta}(\Lambda) - S_{ss}(\Lambda)\, d\Lambda & \\ 0 & \left\{\Lambda : S_{ee}^{\theta}(\Lambda) \le \theta \text{ and } S_{ee}^{\tilde{\theta}}(\Lambda) \le \tilde{\theta}\right\}. \end{cases} \tag{2.103}$$

Except for the condition $\tilde{\theta} > \theta$, (2.102) has exactly the same form as (2.69). Consequently, $\Delta R_{\text{II}}^{\theta,\tilde{\theta}}$ has the same form as (2.71). For completeness, (2.71) is repeated here and we note it is valid for $\tilde{\theta} > \theta$:

$$\Delta R_{\text{II}}^{\theta,\tilde{\theta}} = \begin{cases} \dfrac{1}{8\pi^2}\displaystyle\iint_{\Lambda} \log_2 \dfrac{S_{ee}^{\theta}(\Lambda)}{S_{ee}^{\tilde{\theta}}(\Lambda)}\, d\Lambda & \left\{\Lambda : S_{ee}^{\theta}(\Lambda) > \tilde{\theta} \text{ and } S_{ee}^{\tilde{\theta}}(\Lambda) > \tilde{\theta}\right\} \\ \dfrac{1}{8\pi^2}\displaystyle\iint_{\Lambda} \log_2 \dfrac{S_{ee}^{\theta}(\Lambda)}{\tilde{\theta}}\, d\Lambda & \left\{\Lambda : S_{ee}^{\theta}(\Lambda) > \tilde{\theta} \text{ and } S_{ee}^{\tilde{\theta}}(\Lambda) \le \tilde{\theta}\right\} \\ -\dfrac{1}{8\pi^2}\displaystyle\iint_{\Lambda} \log_2 \dfrac{S_{ee}^{\tilde{\theta}}(\Lambda)}{\tilde{\theta}}\, d\Lambda & \left\{\Lambda : S_{ee}^{\theta}(\Lambda) \le \tilde{\theta} \text{ and } S_{ee}^{\tilde{\theta}}(\Lambda) > \tilde{\theta}\right\} \\ 0 & \left\{\Lambda : S_{ee}^{\theta}(\Lambda) \le \tilde{\theta} \text{ and } S_{ee}^{\tilde{\theta}}(\Lambda) \le \tilde{\theta}\right\}. \end{cases} \tag{2.104}$$

Since (2.104) has exactly the same form as (2.71) with the exception that $\tilde{\theta} > \theta$, it is clear under proper motion-compensated prediction that in general $\Delta R_{\text{II}}^{\theta,\tilde{\theta}} \le 0$, and consequently, and somewhat counter intuitively, indicates a potential *improvement* over a non-scalable system. There is an additional factor, however: for the first case in (2.103) $\Delta D_{\text{II}}^{\theta,\tilde{\theta}}$ is not zero as before, but could be significantly high. Interestingly,

the increased distortion is entirely dependent on the form of the motion compensation and not on the form of the power spectral densities for those regions of Λ. These issues are more fully examined in Section 2.4.

2.4 Rate-distortion functions using approximations to S_{ee}^{θ}

As noted in Section 2.2.4, we can make an approximation to $S_{ee}^{\theta}(\Lambda)$ as long as the transition between $S_{ee}^{\theta}(\Lambda) \leq \theta$ and $S_{ee}^{\theta}(\Lambda) \gg \theta$ doesn't cover a wide frequency range. The importance of using (2.55) is that we may now convert the cases which depend on $S_{ee}^{\theta}(\Lambda)$ to cases which depend only on $S_{ss}(\Lambda)$. Ultimately, with this approximation we can find a non-iterative solution to the rate-distortion function for both above and below the MCP rate based entirely on the input spectrum, the motion-compensation method, and the Fourier transform of the displacement estimation error.

2.4.1 Case I, $\tilde{\theta} \leq \theta$, Operating above the MCP Rate

Substituting (2.55) into (2.64) yields

$$D_{\mathrm{I}}^{\theta,\tilde{\theta}} = \frac{1}{4\pi^2} \iint_{\Lambda} \min\left[\tilde{\theta}, S_{ss}(\Lambda)\right] \, d\Lambda. \tag{2.105}$$

Similarly, substituting (2.55) into (2.65) yields

$$R_{\mathrm{I}}^{\theta,\tilde{\theta}} = \begin{cases} \dfrac{1}{8\pi^2} \displaystyle\iint_{\Lambda} \log_2 \frac{\max\left[\theta, S_{ee}^{\mathrm{I},\theta}(\Lambda)\right]}{\tilde{\theta}} \, d\Lambda & \{\Lambda : S_{ss}(\Lambda) > \theta\} \\ \dfrac{1}{8\pi^2} \displaystyle\iint_{\Lambda} \max\left[0, \log_2 \frac{S_{ss}(\Lambda)}{\tilde{\theta}}\right] \, d\Lambda & \{\Lambda : S_{ss}(\Lambda) \leq \theta\}. \end{cases} \tag{2.106}$$

Note for $S_{ee}^{\mathrm{I},\theta}(\Lambda)$ given by either (2.61) or (2.63), (2.106) is entirely a function of the variables θ and $\tilde{\theta}$, and the functions $S_{ss}(\Lambda)$ and $P(\Lambda)$.

In a similar manner, we may simplify (2.70) and (2.71). We observe the following: if $S_{ee}^{\mathrm{appr},\tilde{\theta}}(\Lambda) \leq \tilde{\theta}$, then $S_{ss}(\Lambda) \leq \tilde{\theta} \leq \theta$, which implies $S_{ee}^{\mathrm{appr},\theta}(\Lambda) \leq \tilde{\theta}$. As a consequence the second case in (2.70) cannot occur. Similarly, if $S_{ee}^{\mathrm{appr},\theta}(\Lambda) \leq \tilde{\theta}$, then $S_{ss}(\Lambda) \leq \tilde{\theta}$,

which implies $S_{ee}^{\text{appr},\tilde{\theta}}(\Lambda) \leq \tilde{\theta}$. As a consequence the third case in (2.70) cannot occur. Thus (2.70) becomes

$$\Delta D_{\text{I}}^{\theta,\tilde{\theta}} = 0 \tag{2.107}$$

and

$$\Delta R_{\text{I}}^{\theta,\tilde{\theta}} = \begin{cases} \dfrac{1}{8\pi^2} \displaystyle\iint_{\Lambda} \log_2 \frac{\max\left[\theta, S_{ee}^{\text{I},\theta}(\Lambda)\right]}{\max\left[\tilde{\theta}, S_{ee}^{\text{I},\tilde{\theta}}(\Lambda)\right]}\, d\Lambda & \{\Lambda : S_{ss}(\Lambda) > \theta\} \\ \dfrac{1}{8\pi^2} \displaystyle\iint_{\Lambda} \log_2 \frac{S_{ss}(\Lambda)}{\max\left[\tilde{\theta}, S_{ee}^{\text{I},\tilde{\theta}}(\Lambda)\right]}\, d\Lambda & \left\{\Lambda : \tilde{\theta} < S_{ss}(\Lambda) \leq \theta\right\} \\ 0 & \left\{\Lambda : S_{ss}(\Lambda) \leq \tilde{\theta}\right\}. \end{cases} \tag{2.108}$$

It is interesting to note for $S_{ss}(\Lambda)$ sufficiently large that we may ignore the maximum functions, and the first case in (2.108) reduces for the optimum spatial filter case to

$$\Delta R_{\text{I}}^{\theta,\tilde{\theta}} = \frac{1}{8\pi^2} \iint_{\Lambda} \log_2 \frac{1 - \frac{|P(\Lambda)|^2 S_{ss}(\Lambda)}{S_{ss}(\Lambda)+\theta}}{1 - \frac{|P(\Lambda)|^2 S_{ss}(\Lambda)}{S_{ss}(\Lambda)+\tilde{\theta}}}\, d\Lambda > 0. \tag{2.109}$$

Thus even under optimum conditions there is always some extra rate required to compensate for the missing data in the MCP loop.

2.4.2 Case II, $\tilde{\theta} > \theta$, Operating below the MCP Rate

By substituting (2.55) into (2.96) we find:

$$D_{\text{II}}^{\theta,\tilde{\theta}-\theta} = \begin{cases} \dfrac{1}{4\pi^2} \displaystyle\iint_{\Lambda} \theta + \frac{1}{1-|F(\Lambda)|^2} \\ \qquad \cdot \min\left[\tilde{\theta} - \theta, \max\left[0, S_{ee}^{I,\theta}(\Lambda) - \theta\right]\right] d\Lambda & \{\Lambda : S_{ss}(\Lambda) > \theta\} \\ \dfrac{1}{4\pi^2} \displaystyle\iint_{\Lambda} S_{ss}(\Lambda)\, d\Lambda & \{\Lambda : S_{ss}(\Lambda) \leq \theta\}. \end{cases} \tag{2.110}$$

Similarly, (2.55) into (2.98) produces:

$$R_{\text{II}}^{\theta,\tilde{\theta}-\theta} = \begin{cases} \dfrac{1}{8\pi^2} \displaystyle\iint_{\Lambda} \max\left[0, \log_2 \frac{\max[\theta, S_{ee}^{\text{I},\theta}(\Lambda)]}{\tilde{\theta}}\right] d\Lambda & \{\Lambda : S_{ss}(\Lambda) > \theta\} \\ 0 & \{\Lambda : S_{ss}(\Lambda) \leq \theta\}, \end{cases} \tag{2.111}$$

where $S_{ee}^{\mathrm{I},\theta}(\Lambda)$ is given by (2.63).

Similar to the reasoning in Section 2.4.1, we wish to eliminate as many cases in (2.103) and (2.104) as possible. Using (2.63), it is clear if $\tilde{\theta} > \theta$, then $S_{ee}^{\mathrm{I},\tilde{\theta}}(\Lambda) > S_{ee}^{\mathrm{I},\theta}(\Lambda)$. Thus, if $S_{ee}^{\mathrm{I},\tilde{\theta}}(\Lambda) \leq \tilde{\theta}$, then $S_{ee}^{\mathrm{I},\theta}(\Lambda) \leq \tilde{\theta}$. Then the second case in (2.103) is eliminated. Since $S_{ss}(\Lambda)\left[1 - \frac{|P(\Lambda)|^2 S_{ss}(\Lambda)}{S_{ss}(\Lambda)+\theta}\right] \leq S_{ss}(\Lambda)$, if $S_{ee}^{\mathrm{I},\theta}(\Lambda) \leq \tilde{\theta}$, then $S_{ss}(\Lambda) \leq \tilde{\theta}$, and thus $S_{ee}^{\mathrm{I},\tilde{\theta}}(\Lambda) \leq \tilde{\theta}$. Then the third and fourth cases in (2.103) are eliminated. As a consequence, we can write for $\tilde{\theta} > \theta$

$$\Delta D_{\mathrm{II}}^{\theta,\tilde{\theta}} = \begin{cases} \dfrac{1}{4\pi^2}\displaystyle\iint_{\Lambda} \frac{|F(\Lambda)|^2}{1-|F(\Lambda)|^2}\left(\tilde{\theta}-\theta\right) d\Lambda & \left\{\Lambda : S_{ss}(\Lambda) > \tilde{\theta}\right\} \\ \dfrac{1}{4\pi^2}\displaystyle\iint_{\Lambda} \frac{|F(\Lambda)|^2}{1-|F(\Lambda)|^2}\left(\max[\theta, S_{ee}^{\mathrm{I},\theta}(\Lambda)] - \theta\right) & \left\{\Lambda : \theta < S_{ss}(\Lambda) \leq \tilde{\theta}\right\} \\ \qquad + \max[\theta, S_{ee}^{\mathrm{I},\theta}(\Lambda)] - S_{ss}(\Lambda)\, d\Lambda & \\ 0 & \{\Lambda : S_{ss}(\Lambda) \leq \theta\}\,. \end{cases} \tag{2.112}$$

Similarly, we can eliminate the second and third cases from (2.104) and thus

$$\Delta R_{\mathrm{II}}^{\theta,\tilde{\theta}} = \begin{cases} \dfrac{1}{8\pi^2}\displaystyle\iint_{\Lambda} \log_2 \frac{\max[\theta, S_{ee}^{\mathrm{I},\theta}(\Lambda)]}{\max[\tilde{\theta}, S_{ee}^{\mathrm{I},\tilde{\theta}}(\Lambda)]}\, d\Lambda & \left\{\Lambda : S_{ss}(\Lambda) > \tilde{\theta}\right\} \\ 0 & \left\{\Lambda : S_{ss}(\Lambda) \leq \tilde{\theta}\right\}, \end{cases} \tag{2.113}$$

where $S_{ee}^{\mathrm{I},\theta}(\Lambda)$ is given by (2.55).

It is interesting to note when $S_{ss}(\Lambda)$ is high, $\Delta R_{\mathrm{II}}^{\theta,\tilde{\theta}}$ is negative and weakly a function of $\tilde{\theta}$. On the other hand, $\Delta D_{\mathrm{II}}^{\theta,\tilde{\theta}}$ is a positive affine function of $\tilde{\theta}$, almost certainly overwhelming any gains made with a negative $\Delta R_{\mathrm{II}}^{\theta,\tilde{\theta}}$.

2.5 Optimal Base Rate for Fully Fine Grained Scalable Systems

Here we study an application of the results derived in the previous sections. In the FFGS system, described in Section 2.1, the selection of the base data rate is critical to the performance of the system. If the base rate is set too low, then the gains made from motion-compensated prediction are not realized and the number of bits required

for high quality video is much greater than a non-scalable system. If the base rate is set too high, however, prediction drift becomes a problem at lower rates. Since we have a parametric representation of the change in rate and distortion as compared to a non-scalable system, the parameters which minimize the change will generate the optimum base rate. Let

$$\Delta D^{\theta,\tilde{\theta}} \triangleq \begin{cases} \Delta D_{\mathrm{I}}^{\theta,\tilde{\theta}} & \text{for } \tilde{\theta} \leq \theta \\ \Delta D_{\mathrm{II}}^{\theta,\tilde{\theta}} & \text{for } \tilde{\theta} > \theta \end{cases} \tag{2.114}$$

$$\Delta R^{\theta,\tilde{\theta}} \triangleq \begin{cases} \Delta R_{\mathrm{I}}^{\theta,\tilde{\theta}} & \text{for } \tilde{\theta} \leq \theta \\ \Delta R_{\mathrm{II}}^{\theta,\tilde{\theta}} & \text{for } \tilde{\theta} > \theta. \end{cases} \tag{2.115}$$

There are a number of different ways to perform the optimization. We choose here as a measure the integral of the change in rate and distortion as a function of $\tilde{\theta}$. Then the optimum rate is $R_O^{\theta^*}$, where

$$\theta^* = \underset{\theta_{\min} \leq \theta \leq \theta_{\max}}{\arg\min} \int_{\theta_{\min}}^{\theta_{\max}} \Delta R^{\theta,\tilde{\theta}} + \frac{R_O^{\tilde{\theta}}}{D_O^{\tilde{\theta}}} \Delta D^{\theta,\tilde{\theta}} \, d\tilde{\theta}. \tag{2.116}$$

The constants $\theta_{\min}$ and $\theta_{\max}$ are determined by

$$R_O^{\theta_{\min}} = R_{\max} \text{ and } R_O^{\theta_{\max}} = R_{\min}. \tag{2.117}$$

Evaluation of (2.116) using the results derived in Section 2.4 is given in Section 2.6.

2.6 Evaluation of MCP Scalable Video Rate-Distortion Functions

In this section the results derived in Sections 2.4 and 2.5 are solved numerically using the following input power spectral density [39, 42, 45]:

$$S_{ss}(\Lambda) = S_{ss}(\omega_x, \omega_y) = \begin{cases} \frac{2\pi}{\omega_0^2} \left(1 + \frac{\omega_x^2 + \omega_y^2}{\omega_0^2}\right)^{-3/2} & |\omega_x| \leq \pi f_{sx} \text{ and } |\omega_y| \leq \pi f_{sy} \\ 0 & \text{otherwise.} \end{cases} \tag{2.118}$$

This power spectral density was chosen because it corresponds well to typical values used in the encoding of digital video conference signals at transmission rates of 2 [Mb/s] and below [39].

When $s(\lambda)$ is spatially sampled at the Nyquist rate, f_{sx} and f_{sy} correspond to the horizontal and vertical sampling frequencies, respectively. As the system is assumed to be continuous in the spatial domain, in [39] a sampling format of 360×288 pixels was chosen to allow comparison with practical implementations. Also, ω_0 was selected to be

$$\omega_0 = \frac{\pi f_{sx}}{42.19} = \frac{\pi f_{sy}}{46.15} \tag{2.119}$$

to correspond to a horizontal and vertical correlation of 0.928 and 0.934, respectively, which provides a good match between (2.118) and video signals of this format. Given ω_0, we then choose f_{sx} and f_{sy} such that $f_{sx} f_{sy} = 1$ [pixels/(unit length)^2]. Since wide-sense stationarity is assumed, the following rate-distortion plots measure rate in [bits/pixel]. However, this may easily be converted to bit data rates in [bits/second] by noting if a frame rate of 10 [frames/second] is assumed the above video format produces 1.0368 [megapixels/second] in an uncompressed format. However, it should be noted that deriving this wide-sense stationary approximation from actual video signals is very difficult and thus rate-distortion plots serve more as a guide than actual bound.

In addition to the power spectral density, the displacement estimation error must also be defined. As in [39], a zero mean, Gaussian isotropic probability density function of the following form is used:

$$p_{\Delta d}(\Delta d) = \frac{1}{2\pi\sigma_{\Delta d}^2} \exp\left[-\frac{\Delta d \cdot \Delta d}{2\sigma_{\Delta d}^2}\right], \tag{2.120}$$

and thus,

$$P(\Lambda) = \exp\left[-\frac{\sigma_{\Delta d}^2}{2}\Lambda \cdot \Lambda\right]. \tag{2.121}$$

In Fig. 2.10 are shown four curves with different displacement estimation error variances which essentially reproduce the results in [39] for reference in the figures which follow. (The abscissa and ordinate have been reversed from [39] to conform to the majority of papers which use rate for the abscissa.) For a very accurate estimation such as Curve α with $\sigma_{\Delta d}^2 = 0.04/f_{sx}^2$, the MCP is effective as compared to Curve δ for

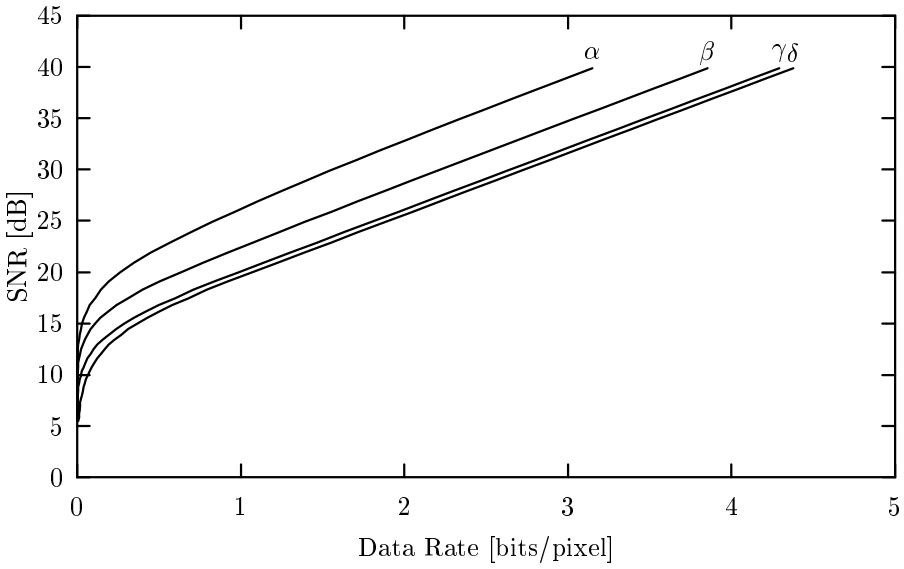


Fig. 2.10. Plot of the rate-distortion functions D_{O}^{θ} and R_{O}^{θ} for optimum MCP non-scalable video codec. Curves α, β, and γ have $\sigma_{\Delta d}^2$ set to $0.04/f_{sx}^2$, $0.15/f_{sx}^2$, and $1.00/f_{sx}^2$ respectively. Curve δ has no motion compensation ($F(\Lambda) = 0$).

which no MCP is applied. As the estimates become less and less exact, for example in Curves β with $\sigma_{\Delta d}^2 = 0.15/f_{sx}^2$ and γ with $\sigma_{\Delta d}^2 = 1.00/f_{sx}^2$, the MCP becomes less effective. Curves α, β, and γ may be thought of as effectively recomputing the required motion vectors for each point on the rate-distortion curve.

Although MCP in real-world sequences has to overcome non-translatory motion, occlusion, block effects, and other non-linear effects, the assumptions made in Section 2.2.4 model the general effects of MCP quite well [42, 45]. To give these results some practical grounding, sequences with low motion are the equivalent of having an accurate prediction of the displacement estimate; conversely, sequences with high motion tend to not have good motion estimates.

Fig. 2.11 shows the effectiveness of encoding above the MCP rate as given in (2.105) and (2.106). In Curve A, the MCP rate is 0.96 [bits/pixel] and intersects the optimum MCP non-scalable curve at this point. The MCP loop receives no more data above this rate, and the motion vectors are exactly the same for each point on the

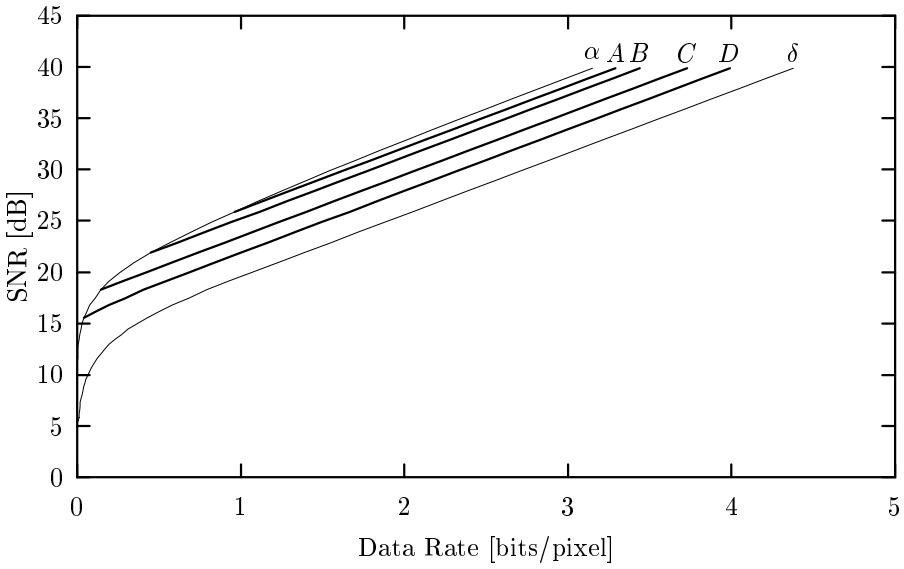


Fig. 2.11. Plot of the rate-distortion functions $D_{\mathrm{I}}^{\theta,\tilde{\theta}}$ and $R_{\mathrm{I}}^{\theta,\tilde{\theta}}$ for $\sigma_{\Delta d}^2 = 0.04/f_{sx}^2$ with various MCP rates. Curves α and δ are repeated from the non-scalable case. For each curve the respective MCP rates in [bits/pixel] are: $R_{\mathrm{MCP}}^A = 0.96$, $R_{\mathrm{MCP}}^B = 0.45$, $R_{\mathrm{MCP}}^C = 0.15$, and $R_{\mathrm{MCP}}^D = 0.04$.

rate-distortion curve. Clearly for Curve A there is some, but not much, loss from scalable coding as compared to Curve α. Conversely, Curve D, with a MCP rate of 0.04 [bits/pixel] exhibits a large increase in distortion (alternatively, requires a large increase in rate) as compared to Curve α. In general, we conclude for this kind of scalability it is preferred to be above the "knee" of the rate-distortion curve so as to take the most advantage of the available MCP coding gain.

If it is assumed a much lower quality displacement estimation is available, as might be the case in a video sequence with large amounts of motion, the results can be characterized as shown in Fig. 2.12. Since the results are now bounded above by Curve β, the absolute loss due to scalability is in general much lower. It is still important, however, to avoid setting the MCP rate too low.

Fig. 2.13 shows the effectiveness of encoding below the MCP rate as given in (2.110) and (2.111). The location of each letter marking the curve indicates the MCP rate;

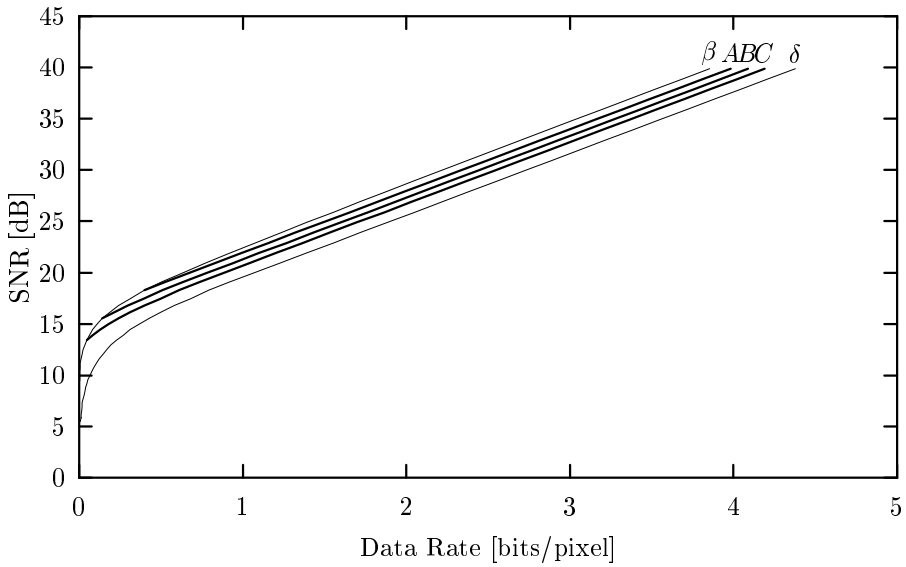


Fig. 2.12. Plot of the rate-distortion functions $D_{\mathrm{I}}^{\theta,\tilde{\theta}}$ and $R_{\mathrm{I}}^{\theta,\tilde{\theta}}$ for $\sigma_{\Delta d}^2 = 0.15/f_{sx}^2$ with various MCP rates. Curves β and δ are repeated from the non-scalable case. For each curve the respective MCP rates in [bits/pixel] are: $R_{\mathrm{MCP}}^A = 0.40$, $R_{\mathrm{MCP}}^B = 0.14$, and $R_{\mathrm{MCP}}^C = 0.05$.

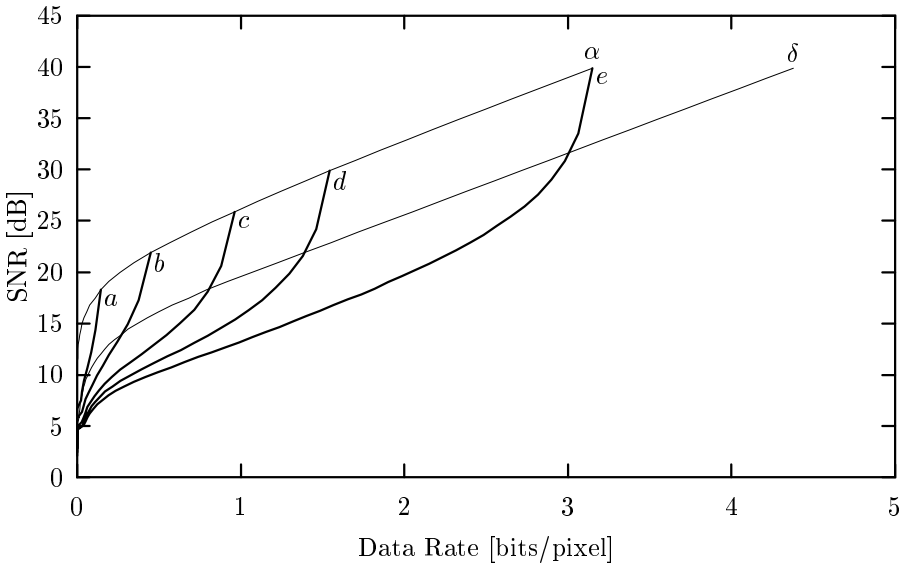


Fig. 2.13. Plot of the rate-distortion functions $D_{\mathrm{II}}^{\theta,\tilde{\theta}-\theta}$ and $R_{\mathrm{II}}^{\theta,\tilde{\theta}-\theta}$ for $\sigma_{\Delta d}^2 = 0.04/f_{sx}^2$ for various MCP rates. Curves α and δ are repeated from the non-scalable case. For each curve the respective MCP rates in [bits/pixel] are: $R_{\mathrm{MCP}}^a = 0.15$, $R_{\mathrm{MCP}}^b = 0.45$, $R_{\mathrm{MCP}}^c = 0.96$, $R_{\mathrm{MCP}}^d = 1.55$, and $R_{\mathrm{MCP}}^e = 3.15$.

below this rate decoding takes place, but the motion vectors remain exactly the same as those obtained at the MCP rate. The first striking feature of these plots is that the decoded version can be significantly lower in SNR than simple intraframe coding for the same data rate. Note if an embedded encoder is used, intraframe coding is also scalable since there is no MCP loop to affect. Secondly, while the initial slope is quite steep, eventually the system stabilizes, albeit at a relatively low SNR. The situation is reversed from the case where we are decoding above the MCP rate, since performance is better the farther *below* the knee the decoding begins.

In Fig. 2.14 shows the results when the variance of the displacement estimates is relatively high. The initial slope of the scalable video is decreased somewhat, and the curves do not show as great a drop from optimal as in Fig. 2.13. Remarkably, however, the non-MCP curve, Curve δ, still does quite well in comparison.

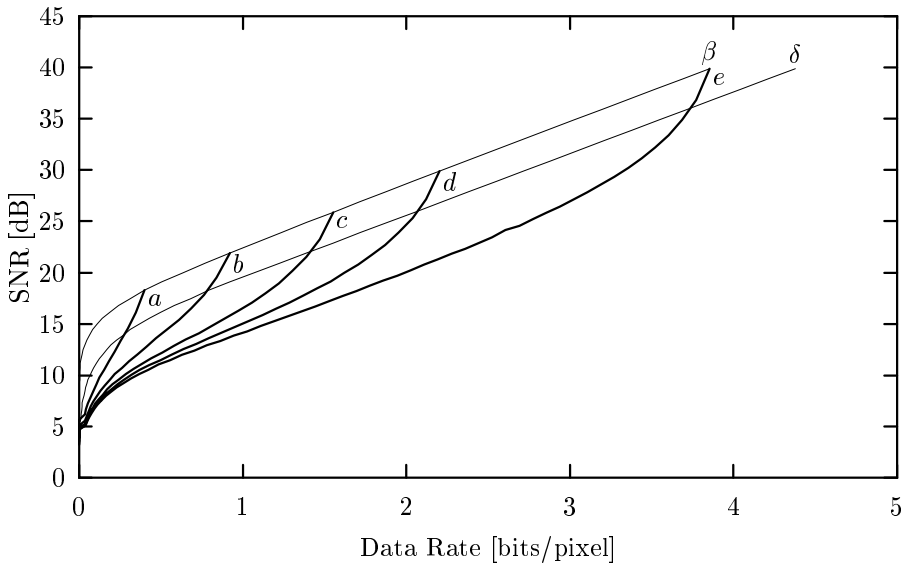


Fig. 2.14. Plot of the rate-distortion functions $D_{\mathrm{II}}^{\theta,\tilde{\theta}-\theta}$ and $R_{\mathrm{II}}^{\theta,\tilde{\theta}-\theta}$ for $\sigma_{\Delta d}^2 = 0.15/f_{sx}^2$ for various MCP rates. Curves β and δ are repeated from the non-scalable case. For each curve the respective base rates in [bits/pixel] are: $R_{\mathrm{MCP}}^a = 0.40$, $R_{\mathrm{MCP}}^b = 0.92$, $R_{\mathrm{MCP}}^c = 1.55$, $R_{\mathrm{MCP}}^d = 2.20$, and $R_{\mathrm{MCP}}^e = 3.85$.

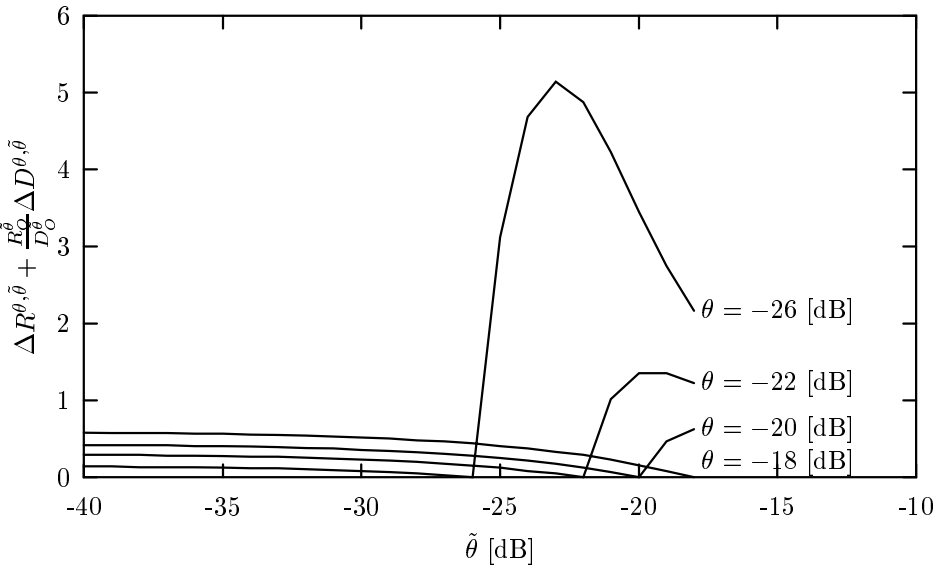


Fig. 2.15. Plot of the function shown in the ordinate for $\sigma^2_{\Delta d} = 0.04/f^2_{sx}$ with $\theta_{\min} = -40$ [dB], $\theta_{\max} = -18$ [dB].

Given the above observations, one can conclude when decoding above the MCP scalable video is very effective if the MCP rate is above the knee in the rate-distortion function. Similarly, the effects of decoding below the MCP rate are ameliorated if the MCP rate is below the knee. These qualitative observations lead to the development of (2.116). From Fig. 2.15 it is clear from the plots of the integrand of (2.116) that the above statements are accurate. For θ below -26 [dB] (roughly 0.9 [bits/pixel]), the error due to decoding below the MCP rate is quite large. Conversely, restricting decoding above the MCP when θ is at -18 [dB] (roughly 0.25 [bits/pixel]) leaves the SNR lagging at the highest rates.

Fig. 2.16 shows a graphical representation to the solution of (2.116) when $\sigma^2_{\Delta d} = 0.04/f^2_{sx}$. Using the measure described by (2.116), θ^* is found to be -20.54 [dB]. The value of the measure at this point is 6.96. When compared to a system which does not decode below the MCP rate, *i.e.,* when $\theta = -18$ [dB] with a measure of 9.63, the technique improves the measure by 27.8 percent.

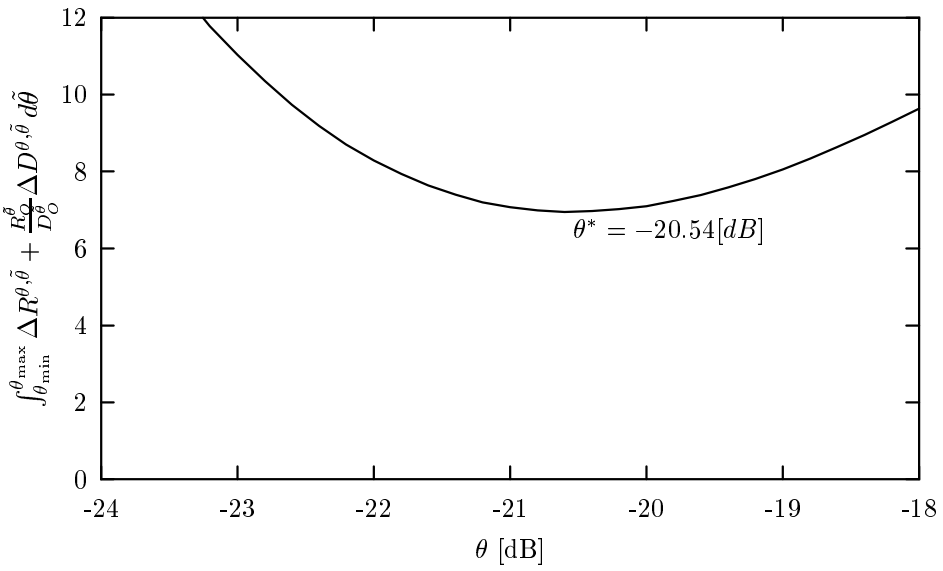


Fig. 2.16. Plot of function shown in the ordinate for $\sigma^2_{\Delta d} = 0.04/f^2_{sx}$ showing the minimum point and thus indicating the optimal base rate for an FGS system. In this case, the minimum corresponds to an optimal base rate of 0.30 [bits/pixel].

2.7 Summary of Results

The most important results developed in the preceding sections are those that describe the performance of a single-loop MCP scalable video codec when compared to an optimum non-scalable video codec operating at with the same generating parameter. This allows comparison of scalable video performance without resorting to actual implementation. The difficulty of numerically simulating these results is roughly similar to simulating the results for the non-scalable case, and thus are computationally tractable.

For the case where we are operating above the MCP rate, we found there is no difference in the distortion when using the same generating parameter, just a

difference in the rates. This is indicated by (2.122) where the difference in distortion is zero and by (2.123) where the differences in rate are positive.

$$\Delta D_{\mathrm{I}}^{\theta,\tilde{\theta}} = 0 \tag{2.122}$$

$$\Delta R_{\mathrm{I}}^{\theta,\tilde{\theta}} = \begin{cases} \dfrac{1}{8\pi^2} \iint_{\Lambda} \log_2 \dfrac{\max\left[\theta, S_{ee}^{\mathrm{I},\theta}(\Lambda)\right]}{\max\left[\tilde{\theta}, S_{ee}^{\mathrm{I},\tilde{\theta}}(\Lambda)\right]} d\Lambda & \{\Lambda : S_{ss}(\Lambda) > \theta\} \\ \dfrac{1}{8\pi^2} \iint_{\Lambda} \log_2 \dfrac{S_{ss}(\Lambda)}{\max\left[\tilde{\theta}, S_{ee}^{\mathrm{I},\tilde{\theta}}(\Lambda)\right]} d\Lambda & \left\{\Lambda : \tilde{\theta} < S_{ss}(\Lambda) \leq \theta\right\} \\ 0 & \left\{\Lambda : S_{ss}(\Lambda) \leq \tilde{\theta}\right\} \end{cases} \tag{2.123}$$

where

$$S_{ee}^{\mathrm{I},\theta}(\Lambda) = S_{ss}(\Lambda)\left[1 - \frac{|P(\Lambda)|^2 S_{ss}(\Lambda)}{S_{ss}(\Lambda) + \theta}\right]. \tag{2.124}$$

Thus, in the case where we are decoding above the MCP rate, it is always possible to attain the same distortion by increasing the rate. In addition, the results also show in the optimum case it is *never* possible to attain the same level of performance with a scalable system as it is with a non-scalable systems, as evidenced by the fact that $\Delta R_{\mathrm{I}}^{\theta,\tilde{\theta}}$ is always positive. These two facts have been shown to be true experimentally, but we believe this is the first time these facts have been proved theoretically.

For the case where we are operating below the MCP rate, we see there is a fundamental difference in (2.125) from (2.122 in that the distortion cannot be made zero, and in fact is a strong function of $\tilde{\theta}$ when compared to the dependence on $\tilde{\theta}$

in (2.126). Interestingly, (2.126) has exactly the same form as as (2.123)—just the operating conditions are different.

$$\Delta D_{\mathrm{II}}^{\theta,\tilde{\theta}} = \begin{cases} \dfrac{1}{4\pi^2}\displaystyle\iint_{\Lambda} \dfrac{|F(\Lambda)|^2}{1-|F(\Lambda)|^2}\left(\tilde{\theta}-\theta\right)\, d\Lambda & \left\{\Lambda : S_{ss}(\Lambda) > \tilde{\theta}\right\} \\ \dfrac{1}{4\pi^2}\displaystyle\iint_{\Lambda} \dfrac{|F(\Lambda)|^2}{1-|F(\Lambda)|^2}\left(\max[\theta, S_{ee}^{\mathrm{I},\theta}(\Lambda)]-\theta\right) & \left\{\Lambda : \theta < S_{ss}(\Lambda) \le \tilde{\theta}\right\} \\ \qquad + \max[\theta, S_{ee}^{\mathrm{I},\theta}(\Lambda)] - S_{ss}(\Lambda)\, d\Lambda & \\ 0 & \{\Lambda : S_{ss}(\Lambda) \le \theta\} \end{cases} \tag{2.125}$$

$$\Delta R_{\mathrm{II}}^{\theta,\tilde{\theta}} = \begin{cases} \dfrac{1}{8\pi^2}\displaystyle\iint_{\Lambda} \log_2 \dfrac{\max[\theta, S_{ee}^{\mathrm{I},\theta}(\Lambda)]}{\max[\tilde{\theta}, S_{ee}^{\mathrm{I},\tilde{\theta}}(\Lambda)]}\, d\Lambda & \left\{\Lambda : S_{ss}(\Lambda) > \tilde{\theta}\right\} \\ 0 & \left\{\Lambda : S_{ss}(\Lambda) \le \tilde{\theta}\right\}, \end{cases} \tag{2.126}$$

where $S_{ee}^{\mathrm{I},\theta}(\Lambda)$ is given above. Clearly the reason this technique is only useful in limited situations is that the distortion rises quite quickly as the rate falls. We believe this is the first time the loss mechanism for prediction drift has been shown theoretically taking into account both distortion and rate effects.

2.8 Comparison to Previously Published Work

In this section we examine the published literature to see if the theory follows actual computed results in real single-loop MCP scalable video systems. In general the computed results are in agreement with the derived theory. Where the results differ, interpretations for the differences are given.

No attempt is made here to assess a quantitative bound using the derived theory on the published results. This is due to several factors. First, it has been noted in [33, 46] that although an optimum rate-distortion analysis model can be developed, the input data might not follow the assumed distribution and the estimate could be off by a significant fraction of the peak signal-to-noise ratio (PSNR). Similarly, in [39] it is noted the bounds developed in Section 2.2.4 represent the upper bound of a rate-distortion function and distributions with identical power spectral densities

but different distributions with the same power spectral density might have better performance. Also, most of the published work uses PSNR which is always higher than the SNR measurement which must be used in the theoretical analysis. Finally, to develop a true bound requires estimates of the input power spectral density and the displacement probability density functions, which in general are difficult to obtain. Nevertheless, the theory developed here still has its usefulness in that it can guide future implementations and separate factors which are due to scalability and those due to coding artifacts.

This section is divided into three areas: video systems which decode above the MCP rate, video systems which decoded below the MCP rate, and finally video systems which decode above and below the MCP rate as needed.

2.8.1 Decoding above the MCP Rate

The most widely recognized method in this class is the MPEG-4 FGS algorithm [10, 12], although a number of other implementations use a similar idea, *e.g.,* [5, 16]. The results presented in [10] are representative of FGS encoding, and are very consistent with the findings in the previous sections. For example, the "Coast-guard" sequence is generally acknowledged to have a high motion content and the FGS implementation only loses 2 [dB] over a scalability (the ratio of the highest to lowest rates) of approximately 9. Conversely the "Carphone" sequence has generally low motion and FGS loses 2 dB over a scalability of 4. Similar results are presented in [12], where the "Stefan" sequence is shown with various base rates. They illustrate the "knee" effect very well, with a loss at 1000 [kb/s] of almost 2 dB for a base rate of 200 [kb/s], and virtually no loss with a base rate of 500 [kb/s]. Also illustrated in the same paper is the case of very high motion. In fact, the authors found FGS *outperformed* the non-scalable codec. This illustrates perfectly an interesting prediction of the theory, since this is possible only if non-optimum motion compensation is used. In effect, the FGS system acts as a low-pass filter which reduces motion artifacts,

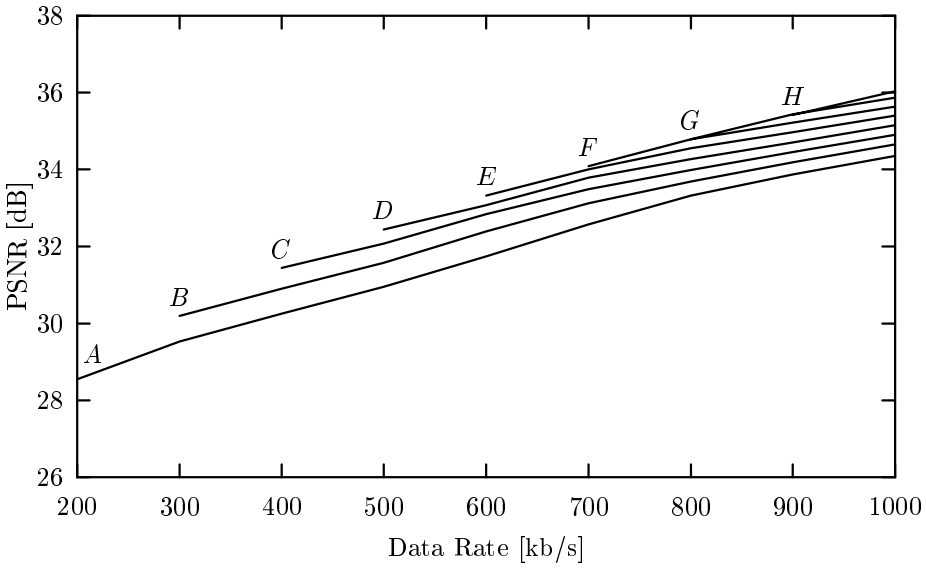


Fig. 2.17. Mean Y-PSNR of MPEG-4 FGS at different MCP rates using the "Coastguard" sequence. The sequence was encoded using the following parameters: a frame size of 352×288 pixels, a frame rate of 10 [frames/s], a GOP size of 15, and a total rate of 1000 [kb/s]. For each curve the respective MCP rates in [kb/s] are: $R^A_{\mathrm{MCP}} = 200$, $R^B_{\mathrm{MCP}} = 300$, $R^C_{\mathrm{MCP}} = 400$, $R^D_{\mathrm{MCP}} = 500$, $R^E_{\mathrm{MCP}} = 600$, $R^F_{\mathrm{MCP}} = 700$, $R^G_{\mathrm{MCP}} = 800$, and $R^H_{\mathrm{MCP}} = 900$.

although in a non-optimum way. The theory is also applicable to other than classic block-MCP DCT systems; we see the same tendencies in [47] , which describes a video compression system using the concept of Matching Pursuits. As the base rate increases the disparity between the non-scalable and scalable system decreases in [47] exactly as predicted.

In Fig. 2.17 are presented operational rate-distortion plots for the "Coastguard" sequence prepared using an implementation of the MPEG-4 FGS algorithm. The results are consistent with the papers quoted above. Note the distinct disadvantage for MPEG-4 FGS when the base rate is below the knee of the optimum rate-distortion curve. The parameters used to derive the rate-distortion functions are also used for Fig. 2.18(b) in Section 2.8.3, and show that even if the compression method is different the same scalability attributes as predicted by the theory still hold.

2.8.2 Decoding Below the MCP Rate

In [31], the author recognizes distortion in a single loop scalable MCP codec operating below the MCP rate is due to quantization and prediction drift, but the author assumes, rather than proves, the two are uncorrelated. Also, the author recognizes that overlapped-block motion compensation, which is equivalent to the Wiener filter described here [48], is an effective method of reducing prediction drift. However, the author does not derive a rate-distortion function but computes the rate distortion based on test sequences. The author's increase of 7 [dB] due to optimizations is consistent with the above theory, although it cannot be shown analytically here because of the undefined values of the integrand in (2.88).

In [32], the effect of prediction drift is clearly illustrated. The author conducted an experiment in which the the quantizer step size for the base layer is kept constant at $Q = 15$ and the enhancement layer quantizer is gradually decreased from $Q = 15$ to $Q = 3$. This is equivalent to increasing the MCP rate while keeping the base layer rate constant. The author found, "Even when an enhancement-layer quantizer of 12 is used, the PSNR has fallen in excess of 3 dB within 12 pictures (*i.e.,* 0.5 s)." Thus the steepness of the initial slope from the MCP rate is confirmed. Also, as the MCP rate increases (Q decreases) the drop in PSNR is dramatic but begins to slow at high MCP rates (low values of Q). This result is clearly predicted in Fig. 2.13 and Fig. 2.14. The authors noted a 7.7 [dB] reductions in PSNR after 24 P-frames, which is consistent with the predicted results.

2.8.3 Decoding above and below the MCP Rate

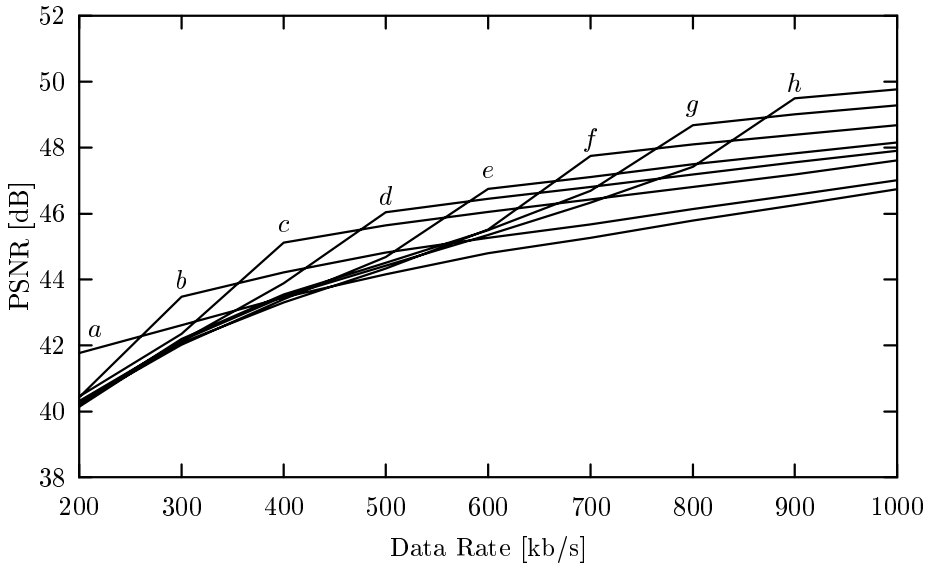
The idea of introducing drift into a MCP FGS-like scalable video codec in order to increase the overall performance is described variously in [18, 4, 19, 20]. The idea was also applied to *SAMCoW* [17] for which no modifications to the algorithm were required [11]. The results from [11] are displayed for reference in Fig. 2.18. The graphs displayed in Fig. 2.18 are equivalent to the combination of Fig. 2.11 and Fig. 2.13

(similarly Fig. 2.12 and Fig. 2.14) where the MCP rates for each curve match. The theory matches actual results well. The prediction of a lower bound when operating below the MCP is well represented, as is the performance above the MCP rate. The theory, however, seems to predict a much higher decrease in SNR when operating below the MCP rate. We believe this is because a GOP size of 15 allows the I-frame to "reset" the PSNR and keep it higher than a case where the GOP size is much larger. We note the theory indicates when operating below the MCP rate it is almost certain to be operating below the non-MCP rate-distortion function, which indicates inserting I-frames would be very effective.

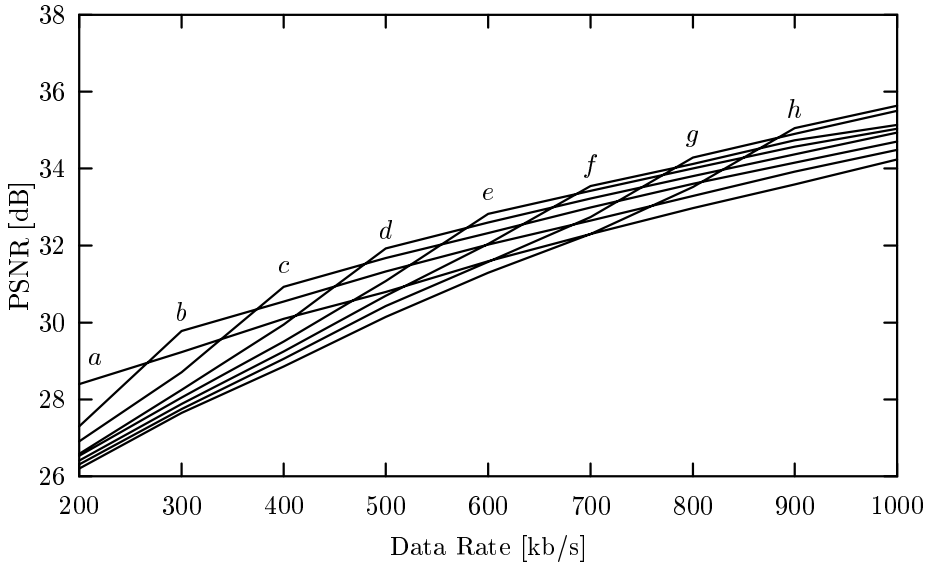
To reduce the effects of drift various authors have introduced improvements to the basic drift procedure. In [19] an additional MCP loop is introduced to better follow the "knee" of the rate distortion curve, and thus the results are much better than indicated by the theory developed here. In [20], the results for decoding below the MCP rate are also much better than indicated by the theory. However, the authors reduce the effects of drift by a feedback method where the number of intraframe updates are increased if the bit stream is sent below the MCP rate. Finally, in [4] the authors use a scheme to sense how much correlation is in the sequence and switch methods to attain the most favorable rate-distortion performance. All of these methods are consistent with the fact that, as predicted by the theory, decoding below the MCP-rate without attention to the rate-distortion attributes may result in very poor performance.

2.9 Conclusions

Presented here was a closed-form expression of the rate-distortion function which serves as a lower bound for all MCP SNR or rate scalable video compression systems. Further insight is gained through deriving these results for fixed translatory motion with uncertainty in the displacement prediction. We found a sufficient condition for stability when decoding below the MCP rate for *any* input power spectral densities and displacement probability density functions. We also showed an application of this



(a)



(b)

Fig. 2.18. Mean Y-PSNR of *SAMCoW* at different MCP rates with two different sequences: (a) "Akiyo" and (b) "Coastguard". The sequences were encoded using the following parameters: a frame size of 352×288 pixels, a frame rate of 10 [frames/s], a GOP size of 15, and a total rate of 1000 [kb/s]. For each curve the respective MCP rates in [kb/s] are: $R^a_{\mathrm{MCP}} = 200$, $R^b_{\mathrm{MCP}} = 300$, $R^c_{\mathrm{MCP}} = 400$, $R^d_{\mathrm{MCP}} = 500$, $R^e_{\mathrm{MCP}} = 600$, $R^f_{\mathrm{MCP}} = 700$, $R^g_{\mathrm{MCP}} = 800$, and $R^h_{\mathrm{MCP}} = 900$.

theory to fully fine-grained scalable systems in determining the optimal base rate. For a given input power spectral density we numerically simulated the rate-distortion function from the derived equations. Finally, we found that these results faithfully predicted the qualitative performance of practical scalable video compression schemes.

2.10 Future Work

There are several interesting topics yet to be explored in the area of modeling scalable video systems. The first is in the area of modeling the input spectrum. As far as this author knows, there is been no further study of determining an input spectrum model which more closely allows agreement with state-of-the-art digital video encoders. Currently, it seems the model mainly overestimates the difficulty of high performance video coding. It might be possible with a more accurate input spectrum to better predict the performance of both non-scalable and scalable encoders. Along these same lines, it may be possible to adopt a different underlying image model of other than a Gaussian distribution. It is clear the elegance of the results would suffer without a forward channel model, but it may still be possible to generate data for, say, a Laplacian distribution using the general concepts, but relying more on computer simulation than mathematical analysis to reach a conclusion and perhaps provide a closer bound for video performance.

Secondly, there has recently been a great deal of interest in using FGS systems for error control. It may be possible to provide rate-distortion plots based on Markovian probability models of loss over a channel. This might be possible if the cascade forward channel model is used where the second optimum codec is itself a stochastic process. If the loss mechanism was able to be tied to the optimum codec, then bounds on performance might be reached where currently only actual implementations are tested on video data.

Another idea to study is the notion of incorporating the effects of intraframes and intraframe updates into the model. As described in the previous section this

technique is used in practical systems to limit the effects of prediction drift. It may be possible to incorporate both intraframe and predicted error frame effects into the same model so the more sophisticated methods of reducing prediction drift described in Section 2.8 can be modeled.

The question of how to incorporate motion vectors into the analysis is an open question. For example, for bi-level scalability it has been proposed by several authors to simply run two prediction loops, one for the lower base level and one for the upper level. A multi-loop model for this kind of system would be easy to generate, but it is unclear how to evaluate it against a single-loop system.

Another interesting question is whether this technique would be useful in the study of 3-D Wavelet encoders. The modeling of the 3-D input spectrum would be a very interesting study and might lead to more efficient application of the wavelet transforms. Further, some 3-D Wavelet encoders use MCP in some form, and this could conveniently be modeled. In essence, 3-D wavelet encoders take advantage of encoding the sub-bands separately; the current analysis could by be modified by applying MCP only to small sub-bands which could be separated out by either layered or cascaded versions of the MSE codecs.

Another interesting topic that might profitably be examined is extending the analysis to multiple description encoders. There have been some studies on the effects of using motion vectors not from the previous frame, but many frames back. By using MSE optimum codecs with this kind of MCP, it might be possible to predict the performance of the perfect multiple description codec. Incorporating an error model as previously specified would make for an interesting comparison paper between FGS and multiple description methods for error control. Currently, there are a number of papers describing actual performance, but no underlying theoretical models.

Finally, so far in this analysis the stochastic filter has been assumed to be exactly the same in the encoder and decoder MCP loops. However, it may not be true that this is optimum in the case of prediction drift. It may be possible to derive an

optimum decoding stochastic filter which minimizes prediction drift in the decoder, assuming some additional information is available about the MCP rate in the encoder.

3. AN INVESTIGATION OF SCALABLE SIMD I/O TECHNIQUES WITH APPLICATION TO PARALLEL JPEG COMPRESSION

3.1 Introduction

In recent years there has been a tremendous increase in the demand for digital imagery. Applications include consumer electronics (Kodak's Photo-CD and HDTV), medical imaging, video-conferencing, scientific visualization, and multimedia. The problem inherent to any digital image or digital video system is the large amount of bandwidth required for transmission or storage. For example, each high resolution Photo-CD image requires 18 megabytes (uncompressed), while HDTV requires a data rate larger than 1.5 gigabits/second (uncompressed). This has driven the research area of image compression to develop algorithms that compress images to lower data rates with better fidelity [49]. One of the ironies of image compression research is that the algorithms which produce these lower data rates are much more computationally complex.

Earlier work examining the mapping of Block Truncation Coding to parallel systems indicated that speedups on the order of the number of processor elements (PEs) in the parallel system were possible [50, 51]. These speedups were indicative of the nonoverlapping block type of structures used in most lossy image and video compression algorithms. Other approaches to decreasing the execution time of compression algorithms have been the use of an array of DSP chips and the use of algorithm and application specific VLSI [52, 53]. Until recently, these methods were the only avenue open for developing real-time image and video processing systems. Parallel computers are very flexible, completely defined in software, and may be programmed in a high-level language [52]. High performance parallel computers are very attractive for

applications where a large amount of imagery is involved. Recently, many parallel computer manufacturers are proposing these systems as video servers because they can compress video data, support serving multiple compressed video data streams and perform the complex operations needed to support a video database, e.g. indexing [54].

In this paper we address the parallel implementation of the JPEG compression and decompression algorithms on the MasPar MP-1, a massively parallel single-instruction multiple-data (SIMD) supercomputer. We chose to implement the algorithms on the MP-1 for a number of reasons. First, the JPEG standard is well known and is used in a variety of applications, including video compression [55, 56] and is prototypical of a large number of block algorithms. Second, block algorithms by their nature require the repeated execution of a single algorithm over the entire array of blocks in an image. This maps extremely well into an SIMD architecture where we are required to have a single program, but may have different data stored in each processor. The JPEG standard requires the use of an 8×8 pixel block as the basic unit of data, which may be easily stored in a single MP-1 processing element (PE). Consequently, because the problem matches so well with the SIMD method of computation we can take advantage of the benefits of a SIMD architecture over multiple-instruction multiple-data architectures. These benefits include less hardware, lower total memory requirements, and simpler communication and synchronization between PEs [57, 58].

In our research we found that the greatest difficulty lies not with the compression algorithm *per se*, but with the input and output problems associated with the parallel architecture. If detail to these problems is ignored any benefit derived from the use of parallelism can be lost. A major focus of this paper is the development of algorithms to address this input/output problem.

In Section 3.2 we describe in detail the JPEG standard in order to define the constraints placed on the parallel compression algorithm. Section 3.3 describes parallel algorithms and the concept of scalability, while Section 3.4 describes the MasPar MP-1. In Section 3.5 we describe the complete parallel JPEG compression algorithm,

including analysis of the parallel output algorithm. The parallel JPEG decompression algorithm is presented in Section 3.6 In Section 3.7 we present a scalability analysis of the algorithm. Finally, experimental results for the implementation of the JPEG and motion JPEG algorithms on the MP-1 are described in Section 3.8

3.2 JPEG Standard

The JPEG Still Picture Compression Standard describes a set of image compression and decompression algorithms for continuous-tone grayscale and color images [59, 60, 61]. There are a number of different options available in the JPEG standard. For example, there are four different modes available for encoding the images: sequential, progressive, lossless, and hierarchical. Also, the JPEG standard specifies two different entropy encoders, specifically, Huffman coding and arithmetic coding. The standard also includes a common baseline algorithm. This algorithm utilizes the discrete cosine transform (DCT) in the sequential mode with a Huffman entropy encoder. This is the algorithm which was implemented on the MP-1, and is described in more detail below.

As shown in Figure 3.1, the baseline (grayscale) compression algorithm has three distinct stages: a DCT stage, a quantization stage, and an entropy binary encoding stage. The sequential color version is similar, except that the RGB color space is converted to the YC_rC_b color space first, and different encoding tables are used for the luminance components and chrominance components. The color components are then interleaved in the compressed data stream.

For the grayscale case, the image data is first scanned in left-to-right, top-to-bottom order, with the pixels grouped into 8×8 nonoverlapping blocks. A two-dimensional DCT is performed on each block, and the DCT coefficients are quantized. The quality factor, a number between 0 and 100, controls the overall resolution of the quantizer and it is set at the time the image is compressed. Finally, the quantized DCT coefficients are Huffman binary encoded.

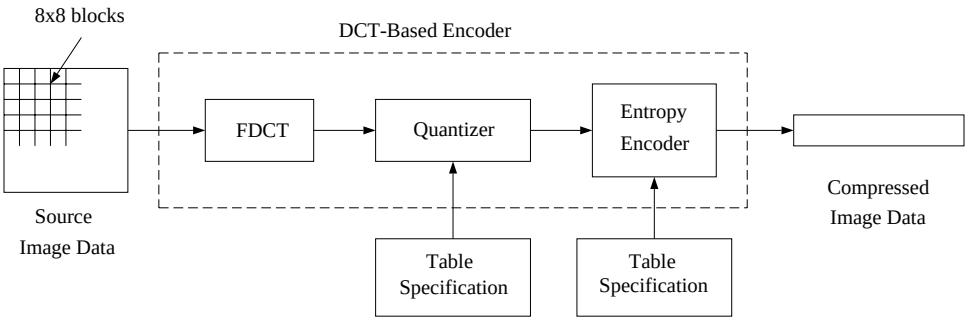


Fig. 3.1. JPEG baseline encoding algorithm.

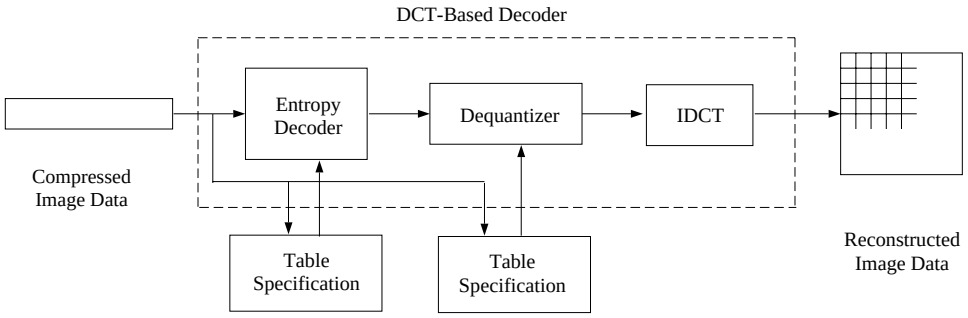


Fig. 3.2. JPEG baseline decoding algorithm.

The various components and tables of the image are separated by a single byte with the value FF_{16} followed by a single byte code. Since the value FF_{16} is also possible in the encoded bit stream, a byte with the value 00_{16} is inserted after all bytes with value FF_{16} which are not separators in the Huffman encoded bit stream. This technique which eliminates false control characters in the data is known as *byte stuffing*.

In Figure 3.2 is shown the baseline (grayscale) decompression algorithm; this algorithm is essentially a reversal of the steps in the compression algorithm. Information for the Huffman decoder and quantizer are carried at the beginning of the compressed data stream.

Motion JPEG is a simple extension to the JPEG standard which allows multiple images, i.e. a video sequence, to be compressed and stored in a single file. Since there is no standard file format for Motion JPEG, we have adopted a file format which closely follows the JFIF format [61]. We assume that images are stored in groups of 32 images with an end-of-image (EOI) marker separating compressed images, and that no change of the values in the quantization or Huffman tables are required for the group of images.

3.3 Parallel Architectures and Algorithms

The price paid for using parallel processing to increase execution speed is an increase in the complexity of developing the algorithm. To offset this disadvantage the parallel algorithm designer can build a parallel algorithm from selected parallel algorithms and techniques which have been found basic to almost all parallel computations [62, 63]. These techniques include partitioning, parallel reduction, parallel prefix computations, pipelining, and pointer jumping [62, 64].

As stated in [65], the scalability of a parallel algorithm on a parallel architecture is a measure of its capability to effectively utilize an increasing number of processors. For analytical purposes we utilize here the notion of *isoefficiency*, which is defined as the rate of change of problem size as a function of the number of processors needed to maintain a fixed processor utilization [58]. As stated in [66], algorithms with isoefficiencies of $O(P \log^c P)$, where P is the number of PEs and c is a small constant, are reasonably scalable for practical purposes. A *scalable* algorithm-architecture will, as a consequence, maintain the same execution time if the problem size/processor size ratio is proportional to the above isoefficiency function.

3.4 The MasPar MP-1

The MasPar MP-1 is a fine-grained massively data-parallel computer. A fully configured system with 16,384 processors can operate at 30 GIPS (peak), with a

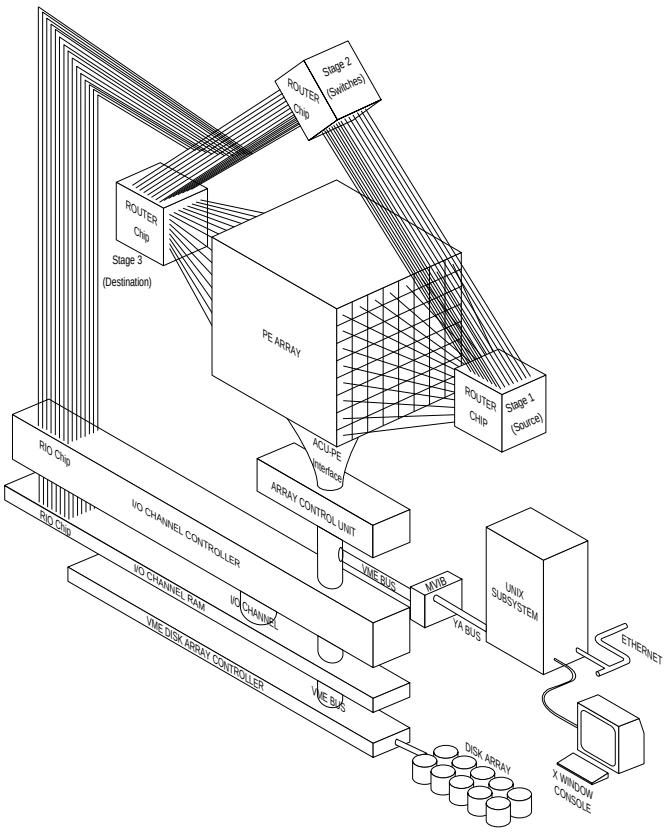


Fig. 3.3. MasPar MP-1 system block diagram.

representative instruction being a 32-bit integer addition. Floating point performance is 1500 MFLOPS single precision (32-bit) and 650 MFLOPS double precision (64-bit) [67]. Figure 3.3 shows the system block diagram of the MasPar [68].

Physically, the unit is divided into two devices, a front end, represented by the UNIX subsystem and X-Window console (Figure 3.3), and the data parallel unit (DPU), which is everything else in Figure 3.3 [68]. The DPU consists of an array control unit (ACU), an array of at least 1024 (16,384 maximum) processing elements (PE), and PE communications mechanisms.

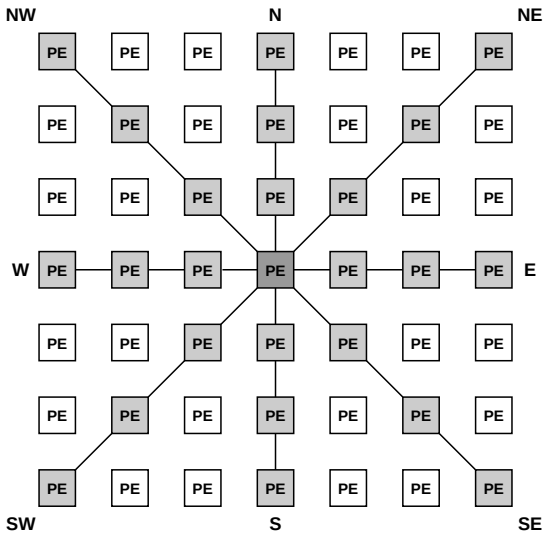


Fig. 3.4. MasPar MP-1 Xnet communications.

The ACU both performs operations on data which does not need to be distributed to the PE array and controls the PE array by sending data and instructions to each PE simultaneously.

The PE array is logically represented by a two dimensional grid, in our case 32×32, 64×64, and 128×128. Each individual PE is a 4-bit load/store arithmetic processing element with dedicated registers and 16 kilobytes of RAM.

There are two communications networks in the DPU: an eight-nearest neighbor network (known as Xnet) and a global router. The Xnet is useful for communicating information which is local to a set of PEs, or to a PE located in a straight line (Figure 3.4), while the global router is mainly used for transmitting data between PEs which are not logically arranged closely together (Figure 3.3).

The programming language for the MP-1 is a parallel variation of C known as MPL [69]. There is a very efficient library of routines for most of the parallel techniques, including `scan`, which executes the parallel prefix and segmented parallel prefix computations, and `reduce` which executes a recursive doubling scheme for any of the associative operators.

Since the MP-1 is a SIMD machine, all of the PEs must execute the same instruction at the same time. There are, however, parallel control structures which allow a PE to become inactive, and not execute the instruction. (Similarly, PEs which actually execute the instruction are termed active.) The PE's local memory can be modified whether the processor is active or inactive.

The MP-1 has a number of routines which allow efficient reading to the PE array and writing from the PE array, including `p_read`, `pp_read`, `p_write`, and `pp_write`. They are similar to the UNIX functions `read` and `write`. The functions `p_read` and `pp_read` differ in that `p_read` reads consecutive blocks of bytes, while `pp_read` may read overlapping blocks or in fact any arbitrary starting position. The functions `p_write` and `pp_write` behave similarly, except that writing overlapping blocks with `pp_write` has an undefined result in the sense that data written by one PE may be incorrectly overwritten by the data in another PE. An important restriction on the functions, however, is that for any single parallel read or parallel write command the number of bytes input or output must be the *same* for all active PEs. As an example, if 9,000 PEs out of 16,384 are active and writing data, then if 4 bytes are to be written from a single PE, a file of length 36,000 bytes is created.

3.5 Parallel JPEG Compression

3.5.1 Core Algorithm

At first glance, the parallel implementation of the JPEG algorithm is straightforward. For example, in a 1024×1024 image and a 128×128 array of PEs, each PE can be assigned an 8×8 block of data. Since the DCT and Quantization steps are completely independent for each 8×8 block, perfect partitioning is achieved and the speedup over a single PE for these two steps is 16,384. Encoding the data using the Huffman binary encoder can also be done independently, except for bit packing the Huffman codewords. Unlike the DCT and quantization steps, the output of the entropy encoder is variable length binary codewords and thus will most likely leave some

of the codewords in a partial byte. The bit packing step is done simply in the serial JPEG algorithm since the number of bits from the previous Huffman encoded block is known. For the parallel implementation, the bit packing step is accomplished by using the same technique as the pointer jumping algorithm, described in Section 3.5.3.

The most difficult part of implementing the JPEG algorithm is not in the algorithm itself, but in realigning the data between the PEs so that the correct operations can be performed with a minimum number of communication steps. The Input Realignment and Output Realignment algorithms, described in Sections 3.5.2 and 3.5.3, respectively, accomplish this task. The two algorithms are not, in fact, inverses of each other but are actually quite different. The Input Realignment algorithm is dependent only on the dimension of the image data with the communication patterns between the PEs being fully deterministic. The Output Realignment algorithm, however, is dependent on the image data itself, specifically the number of bytes in each PE after Huffman binary encoding and JPEG byte stuffing. The communication patterns depend on the encoded image data.

3.5.2 Parallel Input Realignment

As mentioned above, the algorithm presented here is entirely dependent on the size of the input image—thus, routing the data to the proper location can be precomputed. There is a large body of research which has been devoted to studying static permutations on mesh arrays [70, 71], and several researchers have examined the problem on the MP-1 [72, 73, 74]. In [74], the input file was read 8 times (once for each row of an 8×8 block) so that interprocessor communication was minimized at the expense of a higher number of parallel reads. Presented here is an analysis of the required permutation, and a simple parallel algorithm which solves the permutation with a single parallel read.

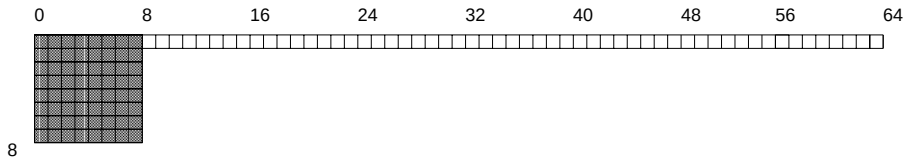


Fig. 3.5. Difference between raster scan information and block information on initial read into PE 0 for a 1024×1024 image.

The basic problem stems from the fact that the data is stored in raster format and the required format is 8×8 blocks. Naturally, if the data in the input image were stored in block format, the input algorithm would be greatly simplified.

Assume that an $n \times n$ pixel image must be read into an array of p PEs, where n mod $8 = 0$. Then to read the entire image, each PE will receive n^2/p bytes (assuming 1 bytes/pixel). The PEs receive the data in raster order, e.g., the first n^2/p bytes go to the first PE. Unfortunately, as illustrated in Figure 3.5, only the first 8 bytes are correct—the next eight bytes must come from the beginning of the second row, which is $n - 8$ bytes away or, in this example, in the next PE.

To make the discussion more specific, we will use the following case: a 1024×1024 pixel image on a 16,384 PE square mesh with dimension 128×128. From above, each PE will contain 64 bytes, exactly one 8×8 block. When the data is first read in, the first 16 PEs will hold the first line of data (1024 bytes); the next 16 PEs will hold the next line, and so on. Consequently, the first row of 128 PEs will hold 8 lines of data, and the second row of 128 PEs will hold the next 8 lines of data. Since the PEs rows do not need to exchange data, each column exchange can be performed in parallel. Unfortunately, the pattern for exchanging columns is not uniform in the sense that each processor sends the same data to the left or right the same number of positions. Hence, each group of 16 processors, which encompass one line of data, are made active, and they transfer their data to the other columns by the proper offsets. The algorithm is shown in Figure 3.6.

ALGORITHM 1.0

Raster to block input realignment

Input: 1-D Array of bytes (known as row-buffer) stored on p^2 processors.

Output: 2-D Array of bytes (known as image-buffer) stored so that a pixel's neighbor exists in the corresponding neighboring PE.

Comment: PE is designated by y (rows) and x (columns), and each PE knows its own y and x coordinates as j and i. Width w of image buffer is exactly $8p$.

RASTER-TO-BLOCK(ROW-BUFFER, IMAGE-BUFFER)
(initialization)
l = width/(number of x processors)
h = height/(number of y processors)
k = (number of x processors)/l
(iteration)
for $r = 0$ **to** $h - 1$
    In parallel,do
        if $rk \leq i < (r+1)k$ **then**
            for $q = 0$ **to** l
                send 8 bytes of data with offset lq from row-buffer to
                PE $(j, il + q)$ with offset r from image-buffer

Fig. 3.6. ALGORITHM 1.0: raster to block input realignment, $w = 8p_x$.

Extension of these results for larger and smaller images is reasonably straightforward. The only required assumption is that the image data width be exactly divisible by 64. Without this assumption, the input image data is loaded across the PEs in a much more inconvenient way and increases greatly the number of communications required and the complexity of the algorithm.

If p_x is the number of columns of the PE array, then for an image width $w \leq 8p_x$ the PEs with a column index of greater than or equal to $w/8$ are made inactive before the data is read into the PE array. As an example, if the image width is 768, then for a 16,384 PE MP-1, the PEs with a column index of 96 or greater are inactive. With the appropriate modifications, Algorithm 1.0 (Figure 3.6) may still be used to redistribute the image data to the proper position.

For an image with width $w > 8p_x$, there are several possible data allocation methods. One method which we have examined is to load the data so that each PE contains $\lceil w/(8p_x) \rceil$ blocks. For example, an image of size 1024×2048 would require two 8×8 blocks in each PE for a 16,384 PE MP-1. The main difficulty with this method is that the memory requirement per PE goes up linearly with the number of image blocks stored in the PE. The method which is currently implemented takes advantage of the fact that once the 8×8 blocks have been formed, the blocks themselves do not need to be arranged so that neighboring blocks are in the neighboring PEs. As an example, an image size of 512×2048 is stored on a 16,384 PE array with PEs of even numbered PE rows holding the left half of the columns of the image (0–1023 indexed pixels) and the PEs of odd numbered PE rows holding the right half of the columns of the image (1024–2047 indexed pixels). A modification must be made to Algorithm 1.0 to account for the fact that two separate lines are contained in the same 8×8 block, but in this case only nearest neighbor communications are required. The modification is shown in Figure 3.7 as Algorithm 1.1. If the total number of 8×8 blocks in the image is greater than the number of active processors, then the input image data is read in sections. As an example, for a 16,384 processor MP-1, an image of size 1024×2048 is read in as two 512×2048 images. Because each section is output separately, a *restart*

marker [61] is inserted between each section of compressed image data. Consequently, the PE memory requirement remains constant for all image sizes.

3.5.3 Parallel Output Realignment

As noted in Section 3.5.1, after the JPEG byte stuffing step each PE contains an array of bytes whose number is dependent on the image data in the associated 8×8 block. Figure 3.8(a) shows a 1024×1024 grayscale image; Figure 3.8(b) is the spatial distribution of the number of bytes in each 8×8 block after Huffman binary encoding. A black pixel indicates 0 bytes of data, while a white pixel indicates the maximum number of bytes in the PE, in this case 24 bytes. (The pixel values have been replicated in a 8×8 block so that the input image and the Huffman encoded magnitudes image are the same size.) Note how the bytes are highly correlated with the original image. As a further complication, we have the restriction that the JPEG compression algorithm requires that the compressed data be stored or transmitted in sequential block order.

The write primitives available for the MP-1 (Section 3.4) do not have the capability of simultaneously writing data which is stored in arrays of varying length. In this paper we will be specifically examining efficiently writing from the PEs to a parallel disk array, however the parallel output algorithms are generally applicable for any output device or channel.

The most obvious solution might be to have each PE write in turn; on the MP-1 this is a prohibitively expensive solution since the MP-1 operates in SIMD mode; when one processor is writing the others must be inactive. The execution time using this technique for a 1024×1024 grayscale image is approximately 50 [s]—this is an order of magnitude larger than the time the algorithm takes on a serial computer, e.g. a Sun SPARC LX (see Section 3.8).

The two algorithms presented below are based on the following objective: while keeping the bytes in sequential order, realign the data so that each PE either has

ALGORITHM 1.1

Raster to block input realignment, rectangular image size

Input: 1-D Array of bytes (known as row-buffer) stored on p^2 processors.

Output: 2-D Array of bytes (known as image-buffer) stored so that a pixel's neighbor exists in the corresponding neighboring PE.

Comment: PE is designated by y (rows) and x (columns), and each PE knows its own y and x coordinates as j and i. Width w of image buffer is exactly $16p$.

RASTER-TO-BLOCK(ROW-BUFFER, IMAGE-BUFFER)

(initialization)

l = width/(number of x processors)

h = height/(number of y processors)

k = (number of x processors)/l

(iteration)

for $r = 0$ **to** $h - 1$

    In parallel,do

        if $rk \leq i < (r+1)k$ **then**

            for $q = 0$ **to** l

                send 8 bytes of data with offset lq from row-buffer to

                PE $(j, il + q)$ with offset r from image-buffer

if j is even **then**

    save first $l/2$ odd lines

    copy first $l/2$ even lines to first $l/2$ lines

    copy first $l/2$ odd lines from j+1 PE to first four lines

else

    copy first $l/2$ odd lines to first $l/2$ lines

    copy saved $l/2$ lines to last $l/2$ lines

Fig. 3.7. ALGORITHM 1.1: raster to block input realignment, $w = 16p_x$.

Fig. 3.8. (a) Top left: original 1024×1024 grayscale image. (b) Top right: spatial distribution of the number of bytes in each 8×8 block after Huffman binary encoding. (c) Bottom left: decompressed JPEG image. (d) Bottom right: spatial distribution of the number of bytes in each 8×8 block after output of pipelining realignment algorithm.

the same number of bytes or has zero bytes of data. This allows the parallel write function on the MP-1 to efficiently move the data to disk.

Pipelining

The pipelining algorithm, shown in Figure 3.9, is based on using a prefix sum computation to determine the effects of moving the data from one PE to a neighboring PE, coupled with a quotient/remainder operation used to determine how the bytes are to be transferred. For this algorithm and the pointer jumping algorithm, a linear array is assumed embedded in the mesh interconnection network, i.e. the first PE of each row is connected to the last PE on the preceding row. By aligning relative to blocks of maximum size, it is guaranteed that the data will only need to travel to the preceding PEs, and not need to travel to succeeding PEs.

An example for $p = 8$ is shown in Table 3.1. The *data* entry indicates the number of bytes in each processor. A prefix sum is then obtained for the number of bytes, as shown in the entry *prefix sum*. The quotient and remainder functions operate only on the data in the individual PEs. (A block size of 8 was chosen to match the size of the block in the pointer jumping algorithm example, although a block size of 6 would have been sufficient.) The PEs determine the start of a block of 8 bytes by subtracting the PEs predecessors quotient values from its own—a 0 indicates the middle of a block and a 1 the start of a block. The bytes are then separated in the marked PEs into two groups: transfer and store. Once this step is accomplished, the algorithm operates in two stages: the first stage moves a single byte from the current PE to the previous PE on the *transfer* array, if a byte exists in the processor. The second step occurs only for marked processors, where the byte received from the next processor, if it exists, is placed on the *store* array. The algorithm iterates until there are no more bytes to transfer. The term pipelining comes from the fact that each time a byte is transferred, the data has moved overall one step closer to the goal.

ALGORITHM 2.0

Data realignment for efficient parallel output using pipelining

Input: Array of bytes stored on p PEs.

Output: Array of bytes stored on a subset of PEs where each PE, except for possibly the last, has the same number of bytes, b.

Comment: The bytes remain in the same sequential order.

```
WRITE-DATA-PIPELINE(L)
(Initialization)
1 Compute the maximum value, b, of bytes
contained in a single PE
2 Compute a prefix sum of the number of bytes
in each PE
3 In parallel do
4     Find and mark PEs with start of b byte blocks
      using a prefix sum quotient with divisor b
5     Find extra bytes in marked PEs using a prefix
      sum remainder with divisor b
(Iteration)
6 In parallel do
7     for i = 1 to b
8         if have bytes
9             then send one byte to previous PE
10        if marked PE and received byte
11            then put byte at end of output array
12    if marked PE
13        then write b bytes of output array
```

Fig. 3.9. ALGORITHM 2.0: data realignment for efficient parallel output using pipelining.

Table 3.1
Pipelining Example

PE	0	1	2	3	4	5	6	7
data	1	3	6	3	1	2	4	4
prefix sum	1	4	10	13	14	16	20	24
quotient base 8	0	0	1	1	1	2	2	3
marked PEs	1	0	1	0	0	1	0	1
remainder base 8	1		2			0		0
Iteration 0 transfer	0	3	4	3	1	2	4	4
store	1		2			0		
Iteration 1 transfer		3	3	3	1	1	4	3
store	2		3			1		
Iteration 2 transfer		3	2	3	1	0	4	2
store	3		4			2		
Iteration 3 transfer		3	1	3	0		4	1
store	4		5			3		
Iteration 4 transfer		3	0	2			4	0
store	5		6			4		
Iteration 5 transfer		2		1			3	
store	6		7			5		
Iteration 6 transfer		1		0			2	
store	7		8			6		
Iteration 7 transfer		0					1	
store	8					7		
Iteration 8 transfer							0	
store						8		

If p is the number of PEs and b is the maximum number of bytes in any single PE, then the time complexity of the algorithm is $O(\log p + b)$, as shown below.

Lines 1 and 2 in Figure 3.9 require $O(\log p)$ [71]. Lines 3–5 require $O(1)$, since only local operations and a single communication is required. Lines 7-11 also require $O(1)$, consequently the total for lines 6-11 is $O(b)$.

Figure 3.8(c) shows the result of executing the pipelining algorithm on the image *man*. After the execution of the algorithm only data of length 32 bytes (white) or 0 bytes (black) remains, except for the last PE which contains a length of 2 bytes.

Pointer Jumping

The implementation of the $O(\log b)$ pointer jumping algorithm must take into account the SIMD nature of the MP-1 in which router operations must also have the same number of bytes for all active PEs. The algorithm is described in Figures 3.10 and 3.11.

An example for $p = 8$ is shown in Table 3.2. The first step is to compute the parallel prefix sum from the number of bytes in each PE. Next, the remainder of the parallel prefix values is obtained using a divisor of 2—in essence we find only those PEs whose parallel prefix sum is odd. If the value of the operation was 1, then the PE moves 1 byte from the succeeding PE to its own memory. Note that for the second iteration it is not necessary to compute the prefix sum again: each PE knows the number of bytes it has received and the number received by its previous neighbor, and therefore has all the necessary information to update the prefix sum. If a PE has zero bytes, then it must be removed from the data transfer step—otherwise, the data transfers cannot occur in parallel. This key point is examined in detail in Section 3.5.4.

An important feature of this algorithm is the fact that if a PE must transfer data, it transfers the same number of bytes as all other PEs which must transfer data; this

ALGORITHM 3.0 (part 1)

Data shuffling for efficient parallel output using pointer jumping.

Input: Array of bytes stored on p PE's.

Output: Array of bytes stored on a subset of PEs where each PE, except for possibly the last, has the same number, b, of bytes.

Comment: The bytes remain in the same sequential order as the input.

WRITE-DATA-POINTER-JUMPING(L)

(initialization)

1. Compute the maximum value, $b_{\max}$, of bytes contained in a single PE
2. Find the maximum block size, b, as the smallest power of two greater than $b_{\max}$.
3. Set the pointer to the next PE to PE+1 and set the pointer to the previous PE to PE-1
4. Set all PEs to active
5. Compute a prefix sum of the number of bytes in each PE

Fig. 3.10. ALGORITHM 3.0: data shuffling for efficient parallel output using pointer jumping (part 1).

ALGORITHM 3.0 (part 2)

(iteration)

6. **In parallel, do**

7. **for** i = 1 **to** log(b) **do**

8. update the prefix sum with remainder of the required number of bytes with divisor 2^i

9. **if** there are no bytes in the buffer **and** the PE is active

10. **then**

11. set the next PEs previous pointer to the previous PE

12. set the previous PEs next pointer to the next PE

13. set PE inactive

14. **if** an active PE

15. **then**

16. determine contributed bytes from previous PE

17. transfer the required bytes from next PE

18. update the prefix sum with the number of bytes transferred

19. **if** an active PE

20. **then** write b bytes of output array

Fig. 3.11. ALGORITHM 3.0: data shuffling for efficient parallel output using pointer jumping (part 2).

Table 3.2
Pointer Jumping Example

PE	0	1	2	3	4	5	6	7
data	1	3	6	3	1	2	4	4
prefix sum	1	4	10	13	14	16	20	24
Iteration 1								
sum update	1	4	10	13	14	16	20	24
remainder base 2	1	0	0	1	0	0	0	0
store	2	2	6	4	0	2	4	4
Iteration 2								
sum update	2	4	10	14		16	20	24
remainder base 4	2	0	2	2		0	0	0
store	4	0	8	4		0	4	4
Iteration 3								
sum update	4		12	16			20	24
remainder base 8	4		4	0			4	0
store	8		8	0			8	0

is a critical condition for SIMD machines where PEs cannot operate independently from one another.

The number of iterations required for the algorithm is $\log(b)$, where b is the maximum number of bytes in any single PE. If we assume the time for the transfer of bytes is linear with the number of bytes, the total time complexity of the algorithm is $O(\log p + b)$, as shown below.

Lines 1 and 5 in Figure 3.10 require $O(\log p)$, while Lines 2–4 require $O(1)$. The number of iterations for Lines 7–18 in Figure 3.11 is $\log b$. All of the lines except Line 17 have time complexity $O(1)$. The time complexity for Lines 7–18, however, is $O(b)$ because of the following: at each of the i steps, at Line 17 a transfer of 2^i bytes must occur. Consequently, this requires $O(\sum_{i=1}^{\log b} 2^i)$, or simply $O(b)$.

Figure 3.12 shows the result of executing the pointer jumping algorithm on the image. The figure shows in sequence the result of moving the data after each iteration. After the execution of the algorithm only data of length 32 bytes (white) or 0 bytes (black) remain, except for the last PE which contains 2 bytes.

3.5.4 Analysis

The bit alignment and pointer jumping algorithms illustrated in Section3.5.3 can be proven to produce the correct result in all cases by use of the theorems given below.

Definition 3.5.1 *Let a be a sequence of bytes with length $|a|$. We define $a_i \oplus a_j$ to be the concatenation of sequence a_i with a_j.*

The operation $\oplus$ is closed and associative, but not commutative.

Definition 3.5.2 *Given an integer y, the function $h(a, y)$ returns the first y bytes of the sequence a. The function $\bar{h}(a, y)$ returns the sequence a without the first y bytes.*

The following theorem is used to prove that irregular sequences of bits may be made less irregular through careful transfer of bits. More precisely, the number of

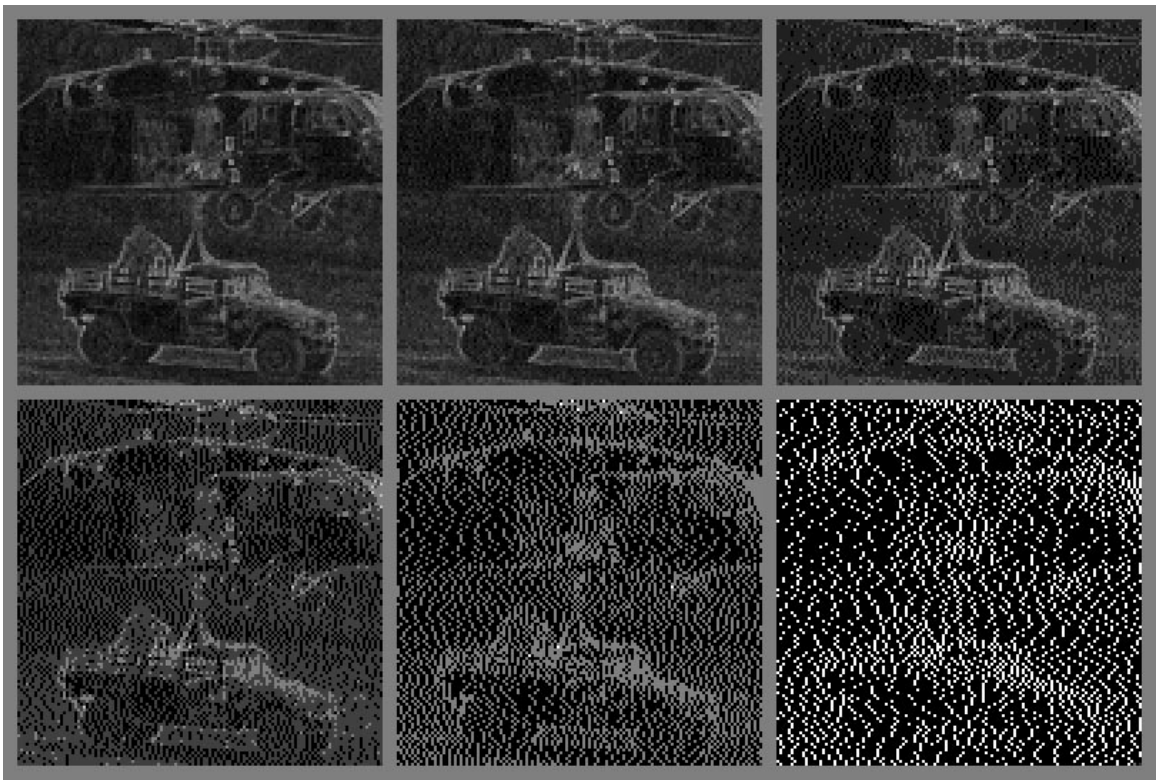


Fig. 3.12. (a) Upper left: spatial distribution of the number of bytes in each 8×8 block after Huffman binary encoding. (b) Upper center: after realignment base 2. (c) Upper right: after after realignment base 4. (d) Lower left: after realignment base 8. (e) Lower center: after realignment base 16. (f) Lower right: after realignment base 32.

bits in any single sequence, once transformed, is exactly a multiple of the modulo base m.

Theorem 3.5.1 *Given modulo base $m \in J$, where $J = \{1, 2, \ldots\}$, $m > 1$, and a sequence of bit sequences $\{a_0, a_1, \ldots, a_{p-1}\}$, let*

$$x_i = (m - |a_i| \bmod m) \bmod m \tag{3.1}$$

and

$$y_i = \left(\sum_{j=0}^{i} x_j \right) \bmod m \tag{3.2}$$

Let

$$a'_i = \begin{cases} a_0 \oplus h(a_1, y_0) & \text{if } i = 0 \\ \bar{h}(a_i, y_{i-1}) \oplus h(a_{i+1}, y_i) & \text{if } 1 \leq i \leq p-2 \\ \bar{h}(a_{p-1}, y_{p-2}) & \text{if } i = p-1 \end{cases} \tag{3.3}$$

If

$$\forall\ 0 \leq i \leq p-2 \quad |a_{i+1}| \geq y_i \tag{3.4}$$

then

$$a'_0 \oplus a'_1 \oplus \cdots \oplus a'_{p-1} = a_0 \oplus a_1 \oplus \cdots \oplus a_{p-1} \tag{3.5}$$

and

$$\forall\ \ 0 \leq i \leq p-2 \quad |a'_i| \bmod m = 0 \tag{3.6}$$

Proof Part 1: We note

$$\begin{aligned} & a'_0 \oplus a'_1 \oplus \cdots \oplus a'_{p-1} \\ &= (a_0 \oplus h(a_1, y_0)) \oplus (\bar{h}(a_1, y_0) \oplus h(a_2, y_1)) \oplus \\ &\qquad \cdots \oplus (\bar{h}(a_{n-3} \oplus h(a_{p-1}, y_{p-2})) \oplus (\bar{h}(a_{p-1}, y_{p-2})) \\ &= a_0 \oplus (h(a_1, y_0)) \oplus \bar{h}(a_1, y_0)) \oplus \\ &\qquad \cdots \oplus (h(a_{p-1}, y_{p-2}) \oplus \bar{h}(a_{p-1}, y_{p-2})) \qquad (3.7) \\ &= a_0 \oplus a_1 \oplus \cdots \oplus a_{p-1} \qquad (3.8) \end{aligned}$$

where Equation 3.7 is true because of associativity and Equation 3.8 is true since $h(a_i, y_{-1}) \oplus \bar{h}(a_i, y_{i-1}) = a_i$ by construction.

Part 2: Note that x_i can alternatively be obtained as

$$x_i = \left\lceil \frac{|a_i|}{m} \right\rceil m - |a_i| \tag{3.9}$$

Also we note the fact that $(b \bmod m - b) \bmod m = 0$.

Thus

$$\begin{aligned} |a_i'| &= |a_i| - y_{i-1} + y_i && (3.10)\\ &= |a_i| - y_{i-1} + (y_{i-1} + x_i) \bmod m && (3.11)\\ &= |a_i| - y_{i-1} && (3.12)\\ &\quad +(y_{i-1} + \left\lceil \frac{|a_i|}{m} \right\rceil m - |a_i|) \bmod m && (3.13)\\ &= (y_{i-1} - |a_i|) \bmod m - (y_{i-1} - |a_i|) && (3.14) \end{aligned}$$

Consequently, $|a_i'| \bmod m = 0$. ■

In practice, the restriction in Equation 3.4 equates to

$$\begin{aligned} \forall\, i \;\; |a_i| &\geq \max_i(y_i) \\ &= m - 1 && (3.15) \end{aligned}$$

although this is more restrictive than necessary.

Equation 3.15 is actually very restrictive and points out the need for the pointer jumping technique. As an example, for a typical 1024×1024 grayscale image the maximum number of bytes in a single compressed 8×8 block is approximately 32. If we wish to form blocks of bytes as outlined in Theorem 3.5.1, then the minimum number of bytes required in each PE to guarantee success is 31.

There are, however, two methods that we can use to relax this restriction without changing the basic algorithm significantly.

To show this we require the following Lemma and Corollary.

Lemma 1 *Given modulo base $m \in J$, $m > 1$, if $\forall\, i \;\; |a_i| \geq m$, then $\forall\, i \;\; |a_i'| \geq m$.*

Proof Since $|a_i| \geq m \geq m-1$ the condition in Equation 3.4 in Theorem 3.5.1 is satisfied. Thus $|a_i'| \bmod m = 0$. We know $|a_i'| = |a_i| - y_i + y_{i-1}$, thus

$$\begin{aligned} |a_i'| &\geq \min_i(|a_i| - y_i + y_{i-1}) && (3.16)\\ &= \min_i(|a_i|) - \max_i(y_i) + \min_i(y_{i-1}) && (3.17)\\ &\geq m - (m-1) + 0 && (3.18)\\ &\geq 1 && (3.19) \end{aligned}$$

Since $|a_i'| \in \{0, m, 2m, \ldots\}$, consequently $|a_i'| \geq m$. ■

Corollary 1 *Given modulo base $m \in J$, $m > 1$, and $n \in J$, if $\forall\, i \;\; |a_i| \geq nm$, then $\forall\, i \;\; |a_i'| \geq nm$.*

Proof Since $nm \geq m$, Theorem 3.5.1 is satisfied. By replacing the value of $\min_i(|a_i|)$ in Equation 3.18 with nm, it follows that $|a_i'| \geq nm$. ■

Given Lemma 1 and Corollary 1, then the lower bound can be relaxed as described below in Theorem 3.5.2. Theorem 3.5.2 indicates that by careful selection of the modulo base value, we can implement the regularizing strategy expressed in the first theorem in multiple stages.

Theorem 3.5.2 *Given modulo base $m_* = m_0 m_1$, $m_0 \geq 2$, $m_1 \geq 2$ and if*

$$\forall\, i \;\; |a_i| \geq \frac{m_0 - 1}{m_0} m_* \qquad (3.20)$$

then using Theorem 3.5.1 twice, once with $m = m_1$ (producing $\{a_0^{'}, a_1^{'}, \ldots, a_{p-1}^{'}\}$) and the second time with $m = m_*$ (producing $\{a_0^{''}, a_1^{''}, \ldots, a_{p-1}^{''}\}$) gives

$$a_0^{''} \oplus a_1^{''} \oplus \cdots \oplus a_{p-1}^{''} = a_0 \oplus a_1 \oplus \cdots \oplus a_{p-1} \qquad (3.21)$$

and

$$\forall \;\; 0 \leq i \leq p-2 \;\; |a_i^{''}| \bmod m_* = 0 \qquad (3.22)$$

Proof Need to show that the condition given by Equation 3.4 or more generally by Equation 3.15 is valid for both stages.

Suppose $\forall\, i \quad |a_i| \geq ((m_0 - 1)/m_0)m_* = (m_0 - 1)m_1$.

Then with $m = m_1$ and Corollary 1, $|a_i^{'}| \geq (m_0 - 1)m_1$, and $|a_i^{'}| \bmod m_1 = 0$.

Now, with $|a_i^{'}| \bmod m_1 = 0$, we can rewrite $\{a_0^{'}, a_1^{'}, \ldots, a_{p-1}^{'}\}$ as $\{b_0, b_1, \ldots, b_{p-1}\}$, where b_i represents groups of sequences of bytes of size m_1, with $|b_i^{'}| = |a_i^{'}|/m_1$. Now, since $m = m_* = m_0 m_1$, we can apply Theorem 3.5.1 on $\{b_0, b_1, \ldots, b_{p-1}\}$ with $m = m_0$ with exactly the same results as applying m_* on $\{a_0^{'}, a_1^{'}, \ldots, a_{p-1}^{'}\}$ as long as Equation 3.15 is true. That is $\{a_0^{''}, a_1^{''}, \ldots, a_{p-1}^{''}\}$ and $\{b_0^{'}, b_1^{'}, \ldots, b_{p-1}^{'}\}$ have exactly the same underlying bit streams. But

$$\begin{aligned} |b_i^{'}| &= \frac{|a_i^{'}|}{m_1} \\ &\geq \frac{(m_0 - 1)m_1}{m_1} \\ &\geq m_0 - 1 \end{aligned} \tag{3.23}$$

and the result is proven. ■

An important conclusion supported by Theorem 3.5.2 is that if the lowest number of bits in a single bit sequence is just half of the maximum number, then the regularizing effect of the algorithm may still be applied.

In the case of the bit alignment algorithm, Theorem 3.5.2 is used directly. Recall that the goal of the bit alignment algorithm is to move the Huffman binary coded data so that no partial bytes of data remain in any processor. If the minimum number of bits was guaranteed to be 7, then Theorem 3.5.1 could be used directly since $m = 8$. Unfortunately, for the baseline coding tables the minimum number of bits in any 8×8 block is 6-bits for the luminance case and is 4-bits for the chrominance cases. Consequently, we use an algorithm based on Theorem 3.5.2 with $m_0 = 2$ and $m_1 = 4$.

For the pointer jumping case, we must use Theorem 3.5.2 $\log b$ times, at each stage eliminating those sequences which have zero number of bytes (in the algorithm this is accomplished by pointer jumping.) The starting point is to first eliminate zero

length arrays, and apply Theorem 3.5.1 with $m = 2$. Now, zero length arrays are again eliminated and we apply Theorem 3.5.2 with $m_0 = 2$ and $m_1 = 2$. (Note in this case and those following that the first stage using m_1 has in effect been accomplished by the previous iteration.) The inequality in Equation 3.20 is satisfied since all of the sequences must have a length of 2 bytes or greater (those of length one cannot exist because of the previous iteration, and those of length zero were eliminated.) The process is repeated, with m_1 doubling at each stage until the desired block size is reached.

3.6 Parallel JPEG Decompression

3.6.1 Core Algorithm

Similar to Section 3.5, the parallel implementation of the JPEG decompression algorithm is straightforward if we assume that the Huffman binary coded data has been distributed properly to the PEs. Since each block of quantized DCT coefficients are encoded independently (except for a differencing operation on the DC coefficient), the decompression algorithm is simply a reversal of the steps taking in the compression stages. Once the image data has been decompressed, a reverse of the parallel algorithm presented in Section 3.5.2 is used to write the data out of the PE array. The difficulty here is in the initial mapping of the compressed image data across the PE array.

3.6.2 Parallel Input Realignment for Encoded Data

When encoded, each 8×8 block of DCT coefficients in the image is represented by a sequence of bits, and the number of bits in the sequence is dependent on the image data. It is not necessary for the start of an encoded block to be on a byte boundary. In the grayscale case, the start of a new block of coefficients can be determined two ways: in the Huffman code itself, where either 64 coefficients or a special end-of-block have

been decoded, or the detection a special sequence of two byte-aligned 8-bit numbers FF_{16} and DX_{16}, where $0 \leq X \leq 7$. The sequence of bytes in the second case is known as a *restart marker*, and its purpose is to allow decompression of an image to continue when the compressed image data is corrupted, and to allow parallel decompression of the image [61]. It is not necessary to decode any of the compressed image data to determine the location of the *restart markers*. The markers may be placed after each Minimum Coded Unit (MCU), which is an 8×8 block for the grayscale case, and three 8×8 blocks in the color case where the chrominant values are not subsampled [61].

The key fact which must be ascertained before the data may be decoded in parallel is the starting position of the first bit in each MCU. Unfortunately, without additional *a priori* information or bit-stream markers, a decoder must search the entire bit-stream *in sequence* to determine these locations, which is equivalent to simply decoding the information serially. This is a direct consequence of Huffman coding the data, since the *location* of a series of bits in relation to the others is as important as the actual values.

One possible solution to the above problem is to store the starting positions, or equivalently the number of bits used to encode each MCU, in the header of the JPEG file. It is permitted by the JPEG standard to design an application-specific marker to store this information [61]. (Any serial JPEG reader will ignore this information and decode the data serially.) When reading the information, the data input algorithm reads the locations of the data first. Then using the `pp_read()` function, and a parallel prefix add operation, the data is read in as overlapping blocks, where the block size is the size of the maximum number of bytes. There are several disadvantages to this method. The first is that auxiliary information which is not part of the JPEG standard must be generated by the encoder; specifically, the number of data bits in each MCU must be stored and written to the compressed image data file. Another disadvantage is that other implementations might not benefit from this extra stored information. A MasPar-specific overlapping read is also required, which may not be available on other massively parallel processor machines. This method would

require the storage of two extra bytes per MCU, thus increasing the data rate of the compressed image by 0.25 bits/pixel.

Instead of relying on auxiliary information not specified in the JPEG standard, we use a different approach which relies only on intrinsic baseline JPEG capability, and thus would be suitable for both a parallel or a serial encoding environment. In this method, a *restart marker* must be inserted by the JPEG encoder between each MCU. Because each marker is byte aligned, there is no need to decode the Huffman coded data, but only to scan for the two byte sequence. Restart markers also allow for editing, selective display, and byte error recovery in the decoding process [61]. Two algorithms are described below, one based on a nonoverlapping parallel read, and the other on an overlapping parallel read. Both utilize the byte realignment algorithms described in Section 3.5.3, and require two stages: a preparation stage, where the data or data pointers are properly positioned, and a reading phase, where the data is read to the correct PE. The parallel data realignment algorithms presented in Section 3.5.3 are used to employ the parallel disk array (or parallel I/O RAM, if available) for use as a fast, scalable, parallel global memory. This method requires the storage of two extra bytes per MCU for the *restart markers*, plus approximately 4 extra bits due to the byte alignment, thus increasing the data rate by 0.31 bits/pixel. The code generated by the parallel JPEG encoding can generate data in this format, as well as encoded data in which the *restart markers* are eliminated.

Parallel Encoded Data Realignment using Nonoverlapping Reads

The nonoverlapping read algorithm, described in Figures 3.13 and 3.14, prepares the data by inserting, in parallel, fill bytes of value FF_{16} so that the distance between the start of each MCU is exactly the same value. Specifically, the encoded image data is first read into the PE's in b_l blocks, where pb_l is greater than or equal to the the number of bytes in the compressed image. This value does not need to be precisely correct, and may be estimated off-line. The *restart markers* are then found,

ALGORITHM 4.0

Preparation step for data realignment for efficient parallel input of irregular sequences using pipelining/pointerjumping algorithms for nonoverlapping data input.

Input: Data stored external to the PE array.

Output: Data stored external to the PE array in temporary file with exactly b bytes between consecutive starts of MCUs.

Comment: The bytes remain in the same sequential order.

PREPARE-DATA-NONOVERLAP()

1 Load data to PEs in b_l blocks into array l

2 **In parallel do**

3 Search for restart interval markers

4 Compute number and location of restart markers

5 find the largest MCU size b using a parallel prefix maximum computation

6 Insert 0xFF before restart markers to fill all MCUs to b bytes

7 Write-Data-{Pipelining,PointerJumping}(l)

Fig. 3.13. ALGORITHM 4.0: preprocessing step for data realignment for efficient parallel input using pipelining/pointerjumping algorithms for nonoverlapping data input.

and their respective locations determined. From this data the largest interval, b, is determined, and the fill bytes are inserted. Note that in general due to the irregular nature of the compressed data, some PEs may contain a number of *restart markers*, while others will have none. At this point the data is realigned and rewritten to the disk using either the pipelining or pointer jumping algorithms. In the reading phase of the algorithm, each PE simply reads b bytes of data.

Given p as the number of PEs and b as the size of the largest MCU, and that a read or write may be performed in $O(b)$, the time complexity of the algorithm is $O(\log p + b^2)$, as shown below.

ALGORITHM 5.0

Data retrieval step for efficient parallel input of irregular sequences using pipelining/pointerjumping algorithms for nonoverlapping reads.

Input: Realigned data with exactly b bytes between blocks stored external to the PE array in a temporary file.

Output: Array of bytes stored in PE array such that each PE has the start of a Huffman coded MCU

Comment: The bytes remain in the same sequential order.

RETRIEVE-DATA-NONOVERLAP()

1. Load data to PEs in b blocks from temporary file

Fig. 3.14. ALGORITHM 5.0: data retrieval step for efficient parallel input using pipelining/pointerjumping for nonoverlapping reads.

Since $b_l \leq b$, Lines 1, 3 and 4 for Algorithm 6.0 in Figure 3.13 require $O(b)$. Line 4 requires $O(b)$ to search within the processors and $O(\log p)$ between the processors, for a total of $O(\log p + b)$. Since there are $O(b)$ reset markers in a single PE, and inserting requires $O(b)$ time, Line 5 requires $O(b^2)$. Finally, Line 7, as determined in Section 3.5.3, requires $O(\log p + b^2)$. Algorithm 6.0 in Figure 3.14 requires $O(b)$ time.

Parallel Encoded Data Realignment using Overlapping Reads

For the algorithms shown in Figures 3.15 and 3.16 we employ the very powerful overlapping read function available on the MP-1 as described in Section 3.4. The first four steps of this algorithm are exactly the same as in Section 3.6.2. Instead of rewriting the data, however, only the number of bytes in each MCU is written. Since these values are scattered unevenly across the PE array, and we wish to write the

ALGORITHM 6.0

Preparation step for Data realignment for efficient parallel input of irregular sequences using pipelining/pointerjumping algorithms for overlapping data input.

Input: Data stored external to the PE array.

Output: Number of storage bytes required for each MCU stored external to PE array in a temporary file.

Comment: The bytes remain in the same sequential order.

PREPARE-DATA-OVERLAP()

1. Load data to PEs in b_l blocks into array l
2. **In parallel do**
3. Search for restart interval markers
4. Compute number and location of restart markers in array m
5. Write-Data-{Pipelining,PointerJumping}(m)

Fig. 3.15. ALGORITHM 6.0: preprocessing step for Data realignment for efficient parallel input using pipelining/pointerjumping for overlapping data input.

data in sequential order, we again apply the output realignment algorithms described in Section 3.5.3. The reading algorithm is slightly more complicated in that the realignment data must be first read in and a parallel prefix add performed to determine the location from the head of the file. A parallel `lseek` command is then performed in each PE, and finally the data is read into the PE array.

Given p as the number of PEs and b as the size of the largest MCU, and that a read or write may be performed in $O(b)$, the time complexity of the algorithm is $O(\log p + b)$, as shown below.

Lines 1, 3 and 4 in Algorithm 6.0 in Figure 3.15 require $O(b)$. Line 5 requires $O(\log p + 1)$. For Algorithm 7.0 in Figure 3.16, Line 1 requires $O(1)$, Line 2 requires $O(\log p)$, and Lines 3 and 4 require $O(b)$.

ALGORITHM 7.0

Data retrieval step for efficient parallel input of irregular sequences using pipelining/pointerjumping algorithms for overlapping reads.

Input: Position data stored external to the PE array in temporary file.

Output: Array of bytes stored in PE array such that each PE has the start of a Huffman coded MCU.

Comment: The bytes remain in the same sequential order.

RETRIEVE-DATA-OVERLAP()

1. Load data to PEs in 1 byte blocks from temporary file
2. Perform a parallel prefix add to find location

of start of MCU from head of file

3. Seek in parallel to proper location
4. Read using overlapping read with byte size b

Fig. 3.16. ALGORITHM 7.0: preprocessing step for Data realignment for efficient parallel input using pipelining/pointerjumping for overlapping data input.

3.7 Scalability Analysis

Below we will show that the entire parallel JPEG algorithm on the MP-1, including I/O, is scalable since its isoefficiency function (see Section 3.3) is $O(p \log p)$.

The core JPEG algorithm has an isoefficiency function of $O(p)$. This is true because the DCT, quantization, and Huffman coding are all performed on 8×8 blocks. As a consequence, the algorithms' complexity remains a linear function of the number of pixels.

The parallel input algorithms may be modeled as having an isoefficiency function of $O(p \log p)$. Even though the MP-1 is primarily a mesh architecture, the global router may be modeled as having a hypercube-like complexity.

As shown in Section 3.5.3, the complexity of the parallel output algorithms are $O(b + \log p)$, again assuming hypercube-like complexity for the global router. With the assumption that b $= O(n/p)$, one can show [58] that the isoefficiency function is $O(p \log p)$.

A similar analysis holds for the JPEG decompression algorithm.

Experimental results, shown in Section 3.8 also indicate that the algorithm is scalable. In this case we have not derived isoefficiency functions from the experimental data, but use the property that the execution time for scalable algorithms remains constant if the ratio between the number of pixels and the number of processors is kept constant.

3.8 Algorithm Performance

Experiments were performed using 8-bit grayscale and 24-bit *RGB* color images for six image sizes: 256×256, 256×512, 512×512, 512×1024, 1024×1024, and 1024×2048. The data rates for these images are given in Table 3.3. A minimum restart indicates that at most one *restart marker* is inserted into the bit stream, while a maximum restart indicates a *restart marker* is inserted after every MCU. Although not required, in the maximum restart case the DC coefficient was coded rather than

Table 3.3
Data Rates for the Test Image in bits/pixel with a JPEG Quality Factor of 75

Image Size	minimum restart		maximum restart	
	Grayscale	Color	Grayscale	Color
256×256	2.129	3.410	2.483	3.830
256×512	1.977	3.085	2.328	3.514
512×512	1.618	2.590	1.976	3.028
512×1024	1.510	2.353	1.867	2.803
1024×1024	1.155	1.880	1.517	2.313
1024×2056	1.096	1.728	1.462	2.178

the difference with the previous MCU DC coefficient. For comparison purposes, execution times for a Sun SPARC LX were obtained using the UNIX `time` command (see Table 3.4). The execution times for the MP-1 were obtained by the MPL function `dpuTimerElapsed()`, which has an overhead of 80 [μs] per measurement. In each case the execution times given in this paper were averaged for ten runs of the algorithm. The algorithm which was implemented on the SPARC LX was developed by the Independent JPEG Group [61], and then modified for execution on the MP-1. Specifically, the algorithm was modified so that the MP-1 executes entirely on the ACU and PE array by storing an 8×8 block of pixels on each PE, and performing the DCT, quantization, and Huffman encoding steps entirely contained within the PE. As a final note, the files generated by the compression and decompression algorithms on the SPARC LX and the MP-1 are identical and are compliant with the JPEG File Interchange Format (JFIF) [61].

Table 3.5 shows the total execution times for a 16,384 PE MP-1, as well as a breakdown for each stage of the JPEG algorithm. The operations are exactly analogous to those performed for the serial algorithm, with the exception of the *bit alignment,*

byte stuffing and *output realignment* operations. The execution time needed for these extra parallel operations is a small fraction of the total parallel execution time. The first four operations, *parse command line, open and creat, initialize file reader,* and *initialize*, are executed mainly on the ACU, with some execution on the PEs for the dynamic allocation of memory. These four operations are necessary to set the various options and memory requirements for each stage of the JPEG algorithm. With the exception of opening the input file and creating the output file, this part of the algorithm has to be executed only once if the image parameters remained fixed for multiple files, as would be the case for an intraframe video coder such as motion JPEG. Execution times given in the *non-initialized total* line of the table show the estimated value for a motion JPEG implementation. It is interesting to note that the three entries in Table 3.5, *DCT, quantize,* and *Huffman encode*, the core of the baseline JPEG compression algorithm comprise less than 15 percent of the total execution time in the single image case.

Table 3.6 shows the results of a motion JPEG implementation (described in Section 3.2) along with the effects of scaling the number of PEs. In the first three lines of the table the ratio of the image size to the processor size is constant, and constitutes exactly one 8×8 block. A graph of the frame rate is shown in Figure 3.17. The executions times and frame rates are reasonably constant which indicate that the parallel implementation of JPEG is scalable (see Section 3.3). Similar results are obtained for the case where ratio of the image size to the number of pixels is 128, or the equivalent of two 8×8 blocks. At the maximum image size of 1024×2048, the approximate size of an HDTV image, the MP-1 is able to compress one color image per second. Table 3.7 and Figure 3.18 illustrate the effectiveness of the MP-1 on a video sequence of images of approximately the same spatial resolution resolution as NTSC video. In fact, if a sequence of 32 256×256 images is input to the MP-1, then both grayscale and color images can be compressed in real time, i.e. greater than 30 frames/second. The data in Table 3.7 was derived by multiplying the frame rate by the number of 256×256 images which could be tessellated into the image.

Table 3.4
Execution Times for a Sun SPARC LX for JPEG Compression and Decompression of the Test Grayscale and Color Images with a Quality Factor of 75

Image Size	Compression		Decompression	
	Grayscale Time [s]	Color Time [s]	Grayscale Time [s]	Color Time [s]
256 × 256	0.4	1.4	0.3	0.9
256 × 512	0.8	2.7	0.7	1.7
512 × 512	1.7	5.5	1.2	3.2
512 × 1024	3.3	11.3	2.3	5.9
1024 × 1024	6.5	22.1	4.3	11.5
1024 × 2056	13.0	44.6	8.2	21.9

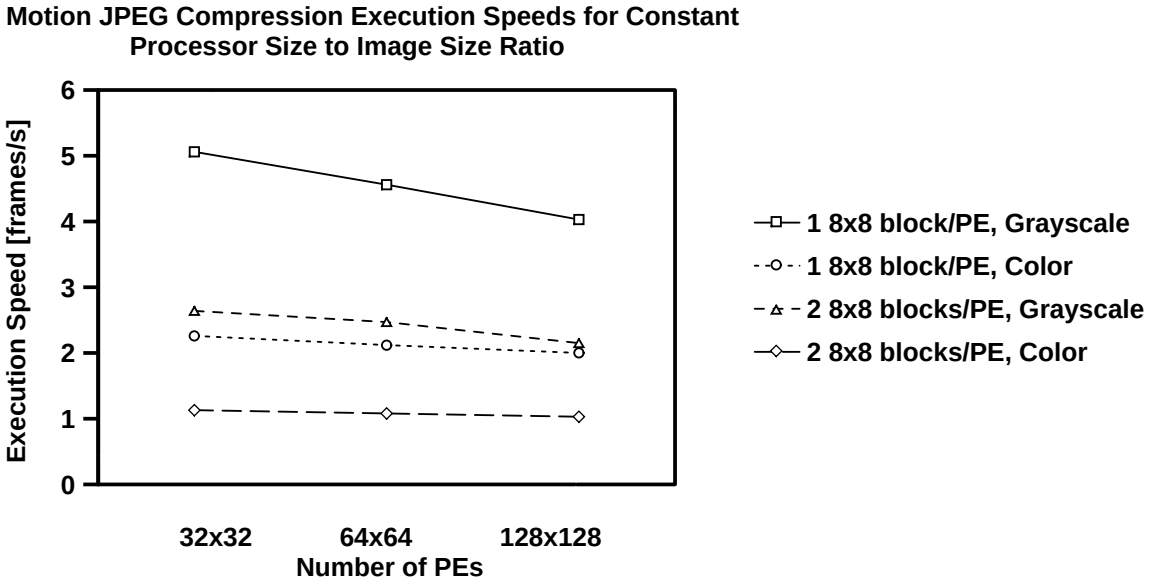


Fig. 3.17. JPEG compression speed in frames per second for constant image size to processor size.

Table 3.5
Execution times for a 16,384 PE MasPar MP-1 for Compressing a 1024×1024 Image Using the Pipelining Algorithm (Writing to the Parallel Disk Array)

Operation	Grayscale Time [s]	Color Time [s]
parse command line	0.0689	0.0685
open and creat	0.0674	0.0663
initialize file reader	0.0178	0.0178
initialize	0.1009	0.1189
write header	0.0613	0.1279
read data	0.0284	0.0626
align data	0.0553	0.1054
color convert/zero mean	0.0085	0.0407
DCT	0.0149	0.0447
quantize	0.0061	0.0183
Huffman encode	0.0583	0.1269
bit alignment	0.0112	0.0145
byte stuffing	0.0163	0.0440
output realignment	0.0098	0.0171
write data	0.0685	0.0802
write trailer/end	0.0043	0.0043
total	0.5979	0.9581
non-initialized total	0.3429	0.6866

Table 3.6
Motion JPEG Compression Execution Times

Processor size	Image size	Number of 8×8 Blocks /Processor	Execution Time [s]		Execution Speed [frame/s]	
			Grayscale	Color	Grayscale	Color
32×32	256×256	1	6.32	14.18	5.06	2.26
64×64	512×512	1	7.02	15.13	4.56	2.12
128×128	1024×1024	1	7.95	16.02	4.03	2.00
32×32	256×512	2	12.12	28.40	2.64	1.13
64×64	512×1024	2	12.95	29.75	2.47	1.08
128×128	1024×2048	2	14.91	31.05	2.15	1.03

Table 3.7
Derived Motion JPEG Compression Execution Times for Constant Image Size

Processor size	Image size	Number of 256×256 Subimages	Execution Speed [frame/s]	
			Grayscale	Color
32×32	256×256	1	5.06	2.26
32×32	256×512	2	5.28	2.26
64×64	256×512	4	18.24	8.48
64×64	512×1024	8	19.76	8.64
128×128	1024×1024	16	64.56	32.00
128×128	1024×2048	32	68.42	33.47

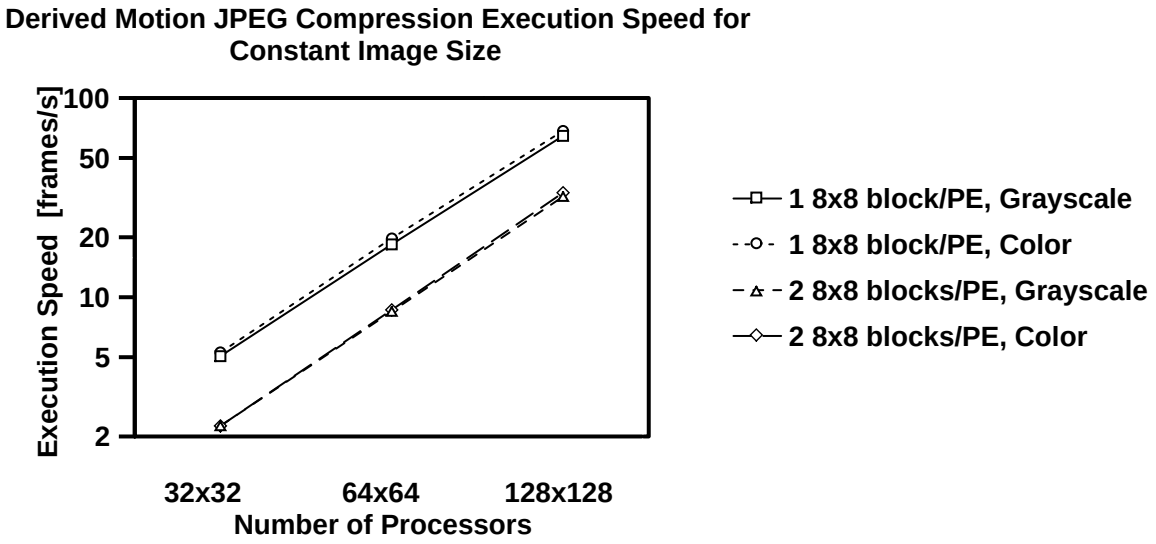


Fig. 3.18. JPEG compression speed in frames per second for constant image size of 256×256.

The execution times discussed above were obtained using the pipelining byte alignment technique which was optimized for a 1024×1024 grayscale image; the execution time for the algorithm was 0.0078 [s]. The execution time for the pointer jumping algorithm using the same image was 0.0249 [s]. We performed a second experiment where the maximum block size, b, was set artificially high—this simulates an 8×8 image block that contains a large number of high frequency coefficients. Because the router setup time remains constant, and the router is more efficient for a large number of bytes, the execution times for the algorithms are virtually identical, 0.0330 [s] for pipelining and 0.0350 [s] for pointer jumping. Thus we conclude the pointer jumping version will be faster in those cases where there are a large number of bytes in a single PE or in the cases where data is not stored at the edge of the PE array, as might be the case for an image which is not exactly divisible into a 128×128 array of 8×8 blocks.

Decompression results for a single 1024×1024 image are shown in Table 3.8. The results are very similar to the compression case, with the single exception that

the Huffman coding section is approximately 4 times higher than the corresponding value shown in Table 3.5. Unlike the encoding algorithm, where several bits may be placed in the output bit stream at once, the decoding algorithm must examine the bit stream one bit at a time, necessitating a high number of decisions for each decoded coefficient. Since the MasPar MP-1 is a SIMD computer, the overhead for keeping a number of the processors idle was quite high in this case. Using the overlapping read algorithm (*alignment* and *read data* in Table 3.8) resulted in data input times of 0.0744 [s] for the grayscale grayscale test image, and 0.0819 [s] for the color test image, both well within the respective times to output the reconstructed RGB image.

Similar to the compression case, we also performed a motion JPEG experiment to test the scalability of the system. The results of this test are presented in Table 3.9 and Figure 3.19, and indicate that the decompression algorithm is also reasonably scalable with respect to a fixed execution time. The times are approximately twice as high as compared to the compression execution times, due to the extra time taken by the Huffman decoding section of the algorithm. Table 3.10 indicates that the MP-1 can decompress a 256×256 grayscale image in real-time, assuming that the file is formatted with *restart markers*. A graph of the data is shown in Figure 3.20.

The execution times discussed above were obtained using the overlapping read alignment technique which was optimized for a 1024×1024 grayscale image; the execution time for the algorithm was 0.0744 [s]. The execution time for the nonoverlapping read using pipelining for the same image was 0.1350 [s] for the preparation phase and a very low 0.00025 [s] for the data reading phase. The use of the pointer jumping algorithm did not significantly change the results. In this case the nonoverlapping read function used the parallel disk array as a global shared memory, but use of an I/O RAM would have decreased the execution time since a significant fraction of time was spent writing the realigned data to disk.

Table 3.8
Execution Times for a 16,384 PE MasPar MP-1 for Decompressing a 1024×1024 Image using the Overlapping Read Algorithm (Writing to the Parallel Disk Array)

Operation	Grayscale Time [s]	Color Time [s]
parse command line	0.0267	0.0274
open and creat	0.2130	0.2130
initialize	0.1391	0.1806
initialize file	0.0527	0.0866
alignment	0.0585	0.0619
read data	0.0164	0.0200
Huffman decode/quant	0.2778	0.4623
DCT	0.0189	0.0563
color convert	0.0016	0.0146
output realignment	0.0533	0.1162
write data	0.0881	0.4033
write trailer/end	0.0012	0.0020
total	0.9473	1.6422
non-initialized total	0.5146	1.1346

Table 3.9
Motion JPEG Decompression Execution Times

Processor size	Image size	Number of 8×8 Blocks /Processor	Execution Time [s]		Execution Speed [frame/s]	
			Grayscale	Color	Grayscale	Color
32×32	256×256	1	13.93	26.90	2.30	1.19
64×64	512×512	1	15.14	29.12	2.11	1.10
128×128	1024×1024	1	17.02	35.97	1.88	0.89
32×32	256×512	2	26.32	51.28	1.22	0.62
64×64	512×1024	2	28.92	55.54	1.11	0.58
128×128	1024×2048	2	33.42	68.48	0.96	0.47

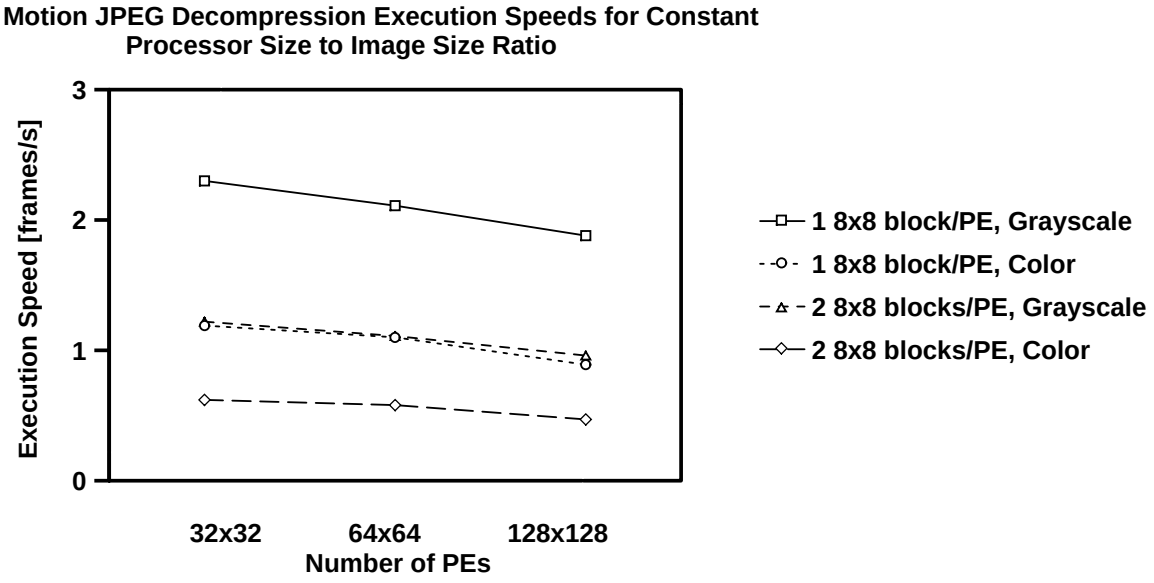


Fig. 3.19. JPEG decompression speed in frames per second for constant image size to processor size.

Table 3.10
Derived Motion JPEG Compression Execution Times for Constant Image Size

Processor size	Image size	Number of 256×256 Subimages	Execution Speed [frame/s] Grayscale	Color
32×32	256×256	1	2.30	1.19
32×32	256×512	2	2.44	1.24
64×64	256×512	4	8.44	4.40
64×64	512×1024	8	8.88	4.64
128×128	1024×1024	16	30.08	14.24
128×128	1024×2048	32	30.72	15.04

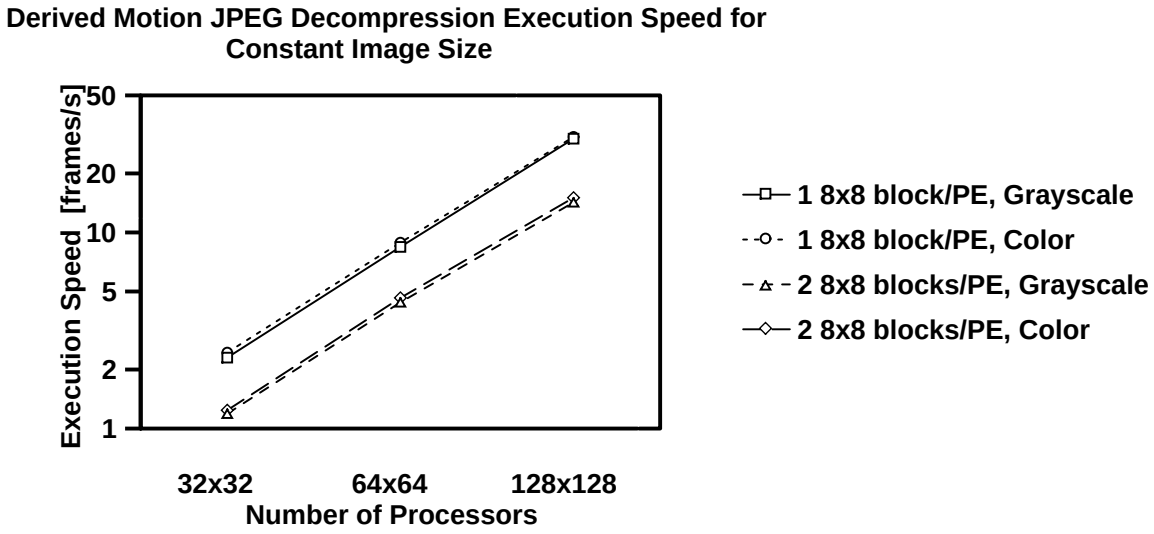


Fig. 3.20. JPEG decompression speed in frames per second for constant image size of 256×256.

3.9 Conclusions

We have described the implementation of the JPEG image compression algorithm on a massively parallel SIMD computer, specifically the MasPar MP-1. Since the JPEG algorithm uses 8×8 nonoverlapping blocks, a partitioning strategy is the obvious choice for a parallel implementation. Implementing the algorithm in parallel is not difficult; the speed bottleneck arises in reading data into the PE array and writing data out of the PE array in such a way that these communication times do not overwhelm the gains obtained by parallel processing.

While the research presented above was developed for a specific purpose, i.e. the encoding of digital images in the JPEG format, the parallel output algorithms can be used in a wider context. The algorithms are bit oriented, so any algorithm which requires input and output of irregular data could use this approach. One very general potential application is the checkpointing of partially processed data. In this case the Huffman coding/output algorithms could be used to checkpoint data much more frequently than might otherwise be possible, since the data is compressed and is quickly written to disk. For data which is stored evenly across the array, the pipelining solution is best. Furthermore, if the data is relatively sparsely scattered through the array in large amounts, then the pointer jumping solution may be more appropriate.

4. SUMMARY

Presented in this dissertation were two different analyses with a common thread of scalablity and video processing. A discussed in Chapter 1, digital video processing is a difficult task because of the high bandwidth and low latency requirements. When coupled with making the algorithms scalable, it becomes an interesting, and useful, problem to analyze.

In Chapter 2, we proved several theoretical properties of single-loop motion compensated prediction rate scalable video codecs. Up to now the properties were well known practical scalable video codecs, but no theoretical analysis had explained the underlying properties completely in the context of rate-distortion theory. Building on the pioneering work of Shannon, the analysis was not aimed at any practical system, but instead found the theoretical best performance given the scalablity constraints. Proven were two useful properties for fine-grained scalable systems. First, these kinds of systems can always attain the same distortion as a non-scalable systems, but require a higher data rate. Secondly, we also proved that if the non-scalable system has optimal motion compensation, then the scalable system will always lag in performance. To put it another way, scalable video systems are always suboptimal with respect to a non-scalable system. For systems which had a deliberate mismatch in the encoder and decoder motion compensated prediction loops due to a reduction in rate, we characterized the dramatic decrease in performance which was well known experiementally, but never characterized in a full rate-distortion sense. Finally, these theories were used to explain why a hybrid systems which use both fine-grained scalablity and prediction drift can perform better than either technique alone.

The work that comprises Chapter 3 was also a theoretical analysis, but centered on the practical task of developing scalable parallel algorithms. In this case the video processing algorithm was the JPEG image compression algorithm, which can be used

for video simply by coding each frame as a JPEG image. The parallel processor was the single-instruction multiple-data massively parallel MasPar MP-1. This unique computer had over 16,000 processors, but compromise to make this many processors work in concert was that each processor had to execute the same instruction at precisely the same time. This proved no difficulty for the basic JPEG compression algorithm because of its block-based nature. However, the most difficult task turned out to be the development of a scalable input/output algorithm. Because of the irregular nature of the compressed data the naïve method of having each processor write in turn was very, very slow—slower even than the excecution of the algorithm on a standard serial processor. By carefully regularizing the data in a scalable way, we were able to develop provably scalable algorithms to process the *locations* of the data. When implemented, these algorithms also had very good performance. Difficulties also arose from the inverse decoding problem. This problem was solved using a technique similar to the compression I/O solution combined with the use of standard JPEG markers to help delineate the block boundaries. The algorithms developed in this chapter are useful in a wider context as well whenever irregular data must be output from the parallel computer. This has potential application to many other video algorithms and even a non-video application such as checkpointing of data in long computations to prevent loss of this data in event of a system failure.

LIST OF REFERENCES

[1] C. E. Shannon, "Coding theorems for a discrete source with a fidelity criterion," *IRE National Convention Record, Part 4*, pp. 142–163, 1959.

[2] A. R. Reibman, Y. Wang, X. Qiu, Z. Jaing, and K. Chawla, "Transmission of multiple description and layered video over an EGPRS wireless network," *Proceedings of the 2000 International Conference on Image Processing*, pp. 136–139, Vancouver, British Columbia, Canada, September 10–12, 2000.

[3] Advanced Television Systems Committee, *ATSC Standard: Digital Television Standard, Revision B, with Amendment 1.* Washington, DC 20006, August 7, 2001. www.atsc.org.

[4] M. van der Schaar and H. Radha, "Adaptive motion-compensation Fine-Granular-Scalability (AMC-FGS) for wireless video," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, no. 6, pp. 360–371, June 2002.

[5] B. G. Haskell, A. Puri, and A. N. Netravali, *Digital Video: An Introduction to MPEG-2.* Chapman and Hall, 1997.

[6] G. Keesman, R. Hellinghuizen, F. Hoeksema, and G. Heideman, "Transcoding of MPEG bitstreams," *Signal Processing: Image Communication*, vol. 8, no. 6, pp. 481–500, September 1996.

[7] A. Vetro and C. W. Chen, "Rate-reduction transcoding design for wireless video streaming," *Proceedings of the 2002 9th IEEE International Conference on Image Processing*, pp. 29–32, Rochester, New York, September 22–25, 2002.

[8] W.-T. Tan and A. Zakhor, "Real-time Internet video using error resilient scalable compression and TCP-friendly transport protocol," *IEEE Transactions on Multimedia*, vol. 1, no. 2, pp. 172–186, June 1999.

[9] A. R. Reibman, H. Jafarkhani, Y. Wang, M. T. Orchard, and R. Puri, "Multiple-description video coding using motion-compensated temporal prediction," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, no. 3, pp. 193–204, March 2002.

[10] W. Li, "Overview of Fine Granularity Scalability in MPEG-4 video standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, no. 3, pp. 301–317, March 2001.

[11] J. Prades-Nebot, G. W. Cook, and E. J. Delp, "Rate control for fully fine-grained scalable video coders," *Visual Communications and Image Processing 2002*, pp. 828–839, San Jose, California, January 21–23, 2002.

[12] H. M. Radha, M. van der Schaar, and Y. Chen, "The MPEG-4 Fine-Grained Scalable video coding method for multimedia streaming over IP," *IEEE Transactions on Multimedia*, vol. 3, no. 1, pp. 53–68, March 2001.

[13] M. van der Schaar and H. M. Radha, "A hybrid temporal-SNR fine-granular scalability for Internet video," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, no. 3, pp. 318–331, March 2001.

[14] J. M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Transactions on Image Processing*, vol. 41, no. 12, pp. 3345–3462, December 1993.

[15] A. Said and W. A. Pearlman, "New, fast and efficient image codec based on set partitioning in hierarchical trees," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, no. 3, pp. 243–250, June 1996.

[16] M. L. Comer, K. Shen, and E. J. Delp, "Rate-scalable video coding using a zerotree wavelet approach," *Proceedings of the Ninth Image and Multidimensional Digital Signal Processing Workshop*, vol. III, pp. 162–163, Belize City, Belize, March 3–6, 1996.

[17] K. Shen and E. J. Delp, "Wavelet based rate scalable video compression," *IEEE Transactions on Circuits and and Systems for Video Technology*, vol. 9, no. 1, pp. 109–122, February 1999.

[18] M. van der Schaar and H. Radha, "Motion compensation based fine-granularity scalability (MC-FGS)," *Contribution to the MPEG-standard*, vol. m6475, October 2000.

[19] A. R. Reibman and L. Bottou, "Managing drift in DCT-based scalable video coding," *Proceedings of the IEEE Data Compression Conference 2001*, pp. 351–360, Snowbird, Utah, March 27–29, 2001.

[20] C. Buchner, T. Stockhammer, D. Marpe, G. Blättermann, and G. Heising, "Progressive texture video coding," *Proceedings of the International Conference on Acoustics, Speech and Signal Processing 2001*, pp. 1813–1816, Salt Lake City, Utah, May 7-11, 2001.

[21] K. Shen, *A Study of Real Time and Rate Scalable Image and Video Compression.* PhD thesis, School of Electrical and Computer Engineering, Purdue University, December 1997.

[22] E. J. Delp, P. Salama, E. Asbun, M. Saenz, and K. Shen, "Rate scalable image and video compression techniques," *Proceedings of the 42nd Midwest Symposium on Circuits and Systems*, pp. 635–638, Las Cruces, New Mexico, August 8–11, 1999.

[23] E. Asbun, P. Salama, K. Shen, and E. J. Delp, "Very low bit rate wavelet-based scalable video compression," *Proceedings of the IEEE International Conference on Image Processing*, pp. 948–952, Chicago, Illinois, October 4–7, 1998.

[24] E. Asbun, P. Salama, and E. J. Delp, "Encoding of predictive error frames in rate scalable video codecs using wavelet shrinkage," *Proceedings of the IEEE International Conference on Image Processing*, Kobe, Japan, October 25–28, 1999.

[25] E. Asbun, P. Salama, and E. J. Delp, "Preprocessing and postprocessing techniques for encoding predictive error frames in rate scalable video codecs," *Proceedings of the 1999 International Workshop on Very Low Bitrate Video Coding*, pp. 148–151, Kobe, Japan, October 29–30, 1999.

[26] E. Asbun, P. Salama, and E. J. Delp, "A rate-distortion approach to wavelet-based encoding of predictive error frames," *Proceedings of the IEEE International Conference on Image Processing*, pp. 832–836, Vancouver, British Columbia, September 10–13, 2000.

[27] E. Asbun, *Improvements in Wavelet-Based Rate Scalable Video Compression.* PhD thesis, School of Electrical and Computer Engineering, Purdue University, December 2000.

[28] K. Shen and E. J. Delp, "Color image compression using an embedded rate scalable approach," *Proceedings of the IEEE International Conference on Image Processing*, vol. III, pp. 34–37, Santa Barbara, California, October 26–29, 1997.

[29] M. Saenz, P. Salama, K. Shen, and E. J. Delp, "An evaluation of color embedded wavelet image compression techniques," *SPIE Conference on Visual Communications and Image Processing'99*, pp. 282–293, San Jose, California, January 23–29 1999.

[30] K. Shen and E. J. Delp, "A control scheme for a data rate scalable video codec," *Proceedings of the IEEE International Conference on Image Processing*, vol. II, pp. 69–72, Lausanne, Switzerland, September 16–19, 1996.

[31] O. Werner, "Drift analysis and drift reduction for multiresolution hybrid video coding," *Signal Processing: Image Communications*, vol. 8, no. 5, pp. 387–409, July 1996.

[32] J. F. Arnold, M. R. Frater, and Y. Wang, "Efficient drift-free signal-to-noise ratio scalability," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 10, no. 1, pp. 70–82, February 2000.

[33] A. Ortega and K. Ramchandran, "Rate-distortion methods for image and video compression," *IEEE Signal Processing Magazine*, vol. 15, no. 6, pp. 23–50, November 1998.

[34] G. J. Sullivan and T. Wiegand, "Rate-distortion optimization for video compression," *IEEE Signal Processing Magazine*, vol. 15, no. 6, pp. 74–90, November 1998.

[35] M. Effros, "Optimal modeling for complex system design," *IEEE Signal Processing Magazine*, vol. 15, no. 6, pp. 51–73, November 1998.

[36] M. Gallant and F. Kossentini, "Rate-distortion optimized layered coding with unequal error protection for robust internet video," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, no. 3, pp. 357–372, March 2001.

[37] K. Ramchandran, A. Ortega, and M. Vetterli, "Bit allocation for dependent quantization with applications to multiresolution and MPEG video coders," *IEEE Transactions on Image Processing*, vol. 3, no. 5, pp. 533–545, September 1994.

[38] F. Kossentini, W. C. Chung, and M. J. T. Smith, "Rate-distortion-constrained subband video coding," *IEEE Transactions on Image Processing*, vol. 8, no. 2, pp. 145–154, February 1999.

[39] B. Girod, "The efficiency of motion-compensating prediction for hybrid coding of video sequences," *IEEE Journal on Selected Areas in Communications*, vol. SAC-5, no. 7, pp. 1140–1154, August 1987.

[40] T. Berger, *Rate Distortion Theory: A Mathematical Basis for Data Compression.* Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1971.

[41] A. Papoulis, *Probability, Random Variables, and Stochastic Processes.* New York: McGraw-Hill, Inc., third ed., 1991.

[42] B. Girod, "Motion-compensating prediction with fractional-pel accuracy," *IEEE Transactions on Communications*, vol. 41, no. 4, pp. 604–612, April 1993.

[43] G. W. Cook, E. Asbun, and E. J. Delp, "An investigation of robust video streaming using a wavelet-based rate scalable codec," *Proceedings of the SPIE Vol 4310 Visual Communications and Image Processing 2001*, vol. 4310, pp. 422–433, San Jose, California, January 24–26, 2001.

[44] S. Wolfram, *Mathematica:A System for Doing Mathematics by Computer.* Reading, Massachusetts: Addison-Wesley Publishing Company, Inc., 1991.

[45] B. Girod, "Efficiency analysis of multihypothesis motion-compensated prediction for video coding," *IEEE Transactions on Image Processing*, vol. 9, no. 2, pp. 173–183, February 2000.

[46] T. M. Cover and J. A. Thomas, *Elements of Information Theory.* New York: Wiley, 1991.

[47] O. Al-Shaykh, E. Miloslavsky, T. Nomura, R. Neff, and A. Zakhor, "Video compression using matching pursuits," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 9, no. 1, pp. 123–143, February 1999.

[48] M. T. Orchard and G. J. Sullivan, "Overlapped block motion compensation: An estimation-theoretic approach," *IEEE Transactions on Image Processing*, vol. 3, no. 5, pp. 693–699, September 1994.

[49] H. A. Peterson and E. J. Delp, "An overview of digital image bandwidth compression," *Journal of Data and Computer Communications*, vol. 2, no. 3, pp. 39–49, 1990.

[50] L. J. Siegel, E. J. Delp, T. N. Mudge, and H. J. Siegel, "Block truncation coding on PASM," *Proceedings of the 19th Annual Allerton Conference on Communication, Control, and Computing*, pp. 891–900, Monticello, Illinois, September 1981.

[51] T. N. Mudge, E. J. Delp, L. J. Siegel, and H. J. Siegel, "Image coding using the multi-microprocessor system PASM," *Proceedings of the IEEE 1982 Pattern Recognition and Image Processing Conference*, pp. 200–205, Las Vegas, Nevada, June 1982.

[52] K. Shen, G. W. Cook, L. H. Jamieson, and E. J. Delp, "An overview of parallel processing approaches to image compression," *SPIE Conference on Image and Video Compression*, vol. 2186, pp. 197–208, San Jose, California, February 1994.

[53] P. Pirsch, N. Demassieux, and W. Gehrke, "VLSI architecture for video compression—a survey," *Proceedings of the IEEE*, vol. 83, no. 2, pp. 220–246, February 1995.

[54] Y.-H. Chang, D. Coggins, D. Pitt, D. Skellern, M. Thapar, and C. Venkatraman, "An open-systems approach to video on demand," *IEEE Communications Magazine*, vol. 32, no. 5, pp. 68–80, May 1994.

[55] G. Cockroft and L. Hourvitz, "NeXTstep: Putting JPEG to multiple uses," *Communications of the ACM*, vol. 34, no. 4, p. 45, April 1991.

[56] R. A. Quinnell, "Image compression: Part 2," *Electronic Design News*, vol. 38, no. 5, pp. 120–126, March 4 1993.

[57] D. A. Patterson and J. L. Hennessy, *Computer Architecture: A Quantitative Approach*. San Mateo, California: Morgan Kaufmann Publishers, Inc., 1990.

[58] V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing: Design and Analysis of Algorithms*. Redwood City, California: The Benjamin/Cummings Publishing Company, Inc., 1994.

[59] G. W. Wallace, "The JPEG still picture compression standard," *Communications of the ACM*, vol. 34, no. 4, pp. 30–44, April 1991.

[60] G. K. Wallace, "The JPEG still picture compression standard," *IEEE Transactions on Consumer Electronics*, vol. 38, no. 1, pp. xviii–xxxiv, February 1992.

[61] W. B. Pennebaker and J. L. Mitchell, *JPEG Still Image Data Compression Standard*. New York: Van Nostrand Reinhold, 1993.

[62] J. JáJá, *An Introduction to Parallel Algorithms*. Reading, Massachusetts: Addison-Wesley Publishing Company, 1992.

[63] L. H. Jamieson, E. J. Delp, C.-C. Wang, and J. Li, "A software environment for parallel computer vision," *IEEE Computer*, vol. 25, no. 2, pp. 73–77, February 1992.

[64] J. H. Reif, ed., *Synthesis of Parallel Algorithms*. San Mateo, California: Morgan Kaufman Publishers, Inc., 1993.

[65] V. Kumar and A. Gupta, "Analyzing scalability of parallel algorithms and architectures," Preprint 92–020, Army High Performance Computing Research Center, University of Minnesota, Minneapolis, MN, January 1992.

[66] G. Karypis and V. Kumar, "Unstructured tree search on SIMD parallel computers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 10, pp. 1057–1072, October 1994.

[67] J. R. Nickolls, "The design of the MasPar MP-1: A cost effective massively parallel computer," *Proceedings of the Thirty-fifth IEEE Computer Society International Conference*, pp. 25–28, San Francisco, California, February 26–March 2, 1990.

[68] T. Blank, "The MasPar MP-1 architecture," *Proceedings of the Thirty-fifth IEEE Computer Society International Conference*, pp. 20–24, San Francisco, California, February 26–March 2 1990.

[69] P. Christy, "Software to support massively parallel computing on the MasPar MP-1," *Proceedings of the Thirty-fifth IEEE Computer Society International Conference*, pp. 29–33, San Francisco, California, February 26–March 2 1990.

[70] F. Annexstein and M. Baumslag, "A unified approach to off-line permutation routing on parallel networks," *Proceedings of the 2nd Annual ACM Symposium on Parallel Algorithms and Architectures*, pp. 398–406, Crete, Greece, July 2–6, 1990.

[71] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: array, trees, hypercubes*. San Mateo California: Morgan Kaufmann Publishers, 1992.

[72] T. Jochem and S. Baluja, "Massively parallel, adaptive, color image processing for autonomous road following," Technical Report CMU-RI-TR-93-10, Carnegie Mellon University, Pittsburgh, Pennsylvania, May 1993.

[73] MasPar Computer Corporation, Sunnyvale, California, *MasPar Data Display Library (MPDDL) Reference Manual*, July 1992.

[74] A. C. P. Loui, A. T. O. Ogielski, and M. L. Liou, "A parallel implementation of the H.261 video coding algorithm," *Proceedings of the IEEE Workshop on Visual Signal Processing and Communications*, pp. 80–85, Raleigh, North Carolina, September 2–3, 1992.

VITA

Gregory W. Cook was born in Lompoc, California, in 1962. He received the B.E.E. (with highest honors) and M.S.E.E. degrees from the Georgia Institute of Technology, Atlanta, in 1984 and 1985, respectively.

From 1985 to 1991, he was a commissioned officer in the United States Air Force, first as a Radar Countermeasures Engineer at the Avionics Laboratory in Dayton, Ohio, and then as an Instructor of Electrical Engineering at the United States Air Force Academy in Colorado Springs, Colorado. He worked as a graduate assistant under a grant from the Defense Advanced Research Projects Agency from 1993-1996, at the C-SPAN Archives in 1997-1998, and from 1999-2001 under a grant from the Indiana Twenty-first Century Research and Technology Fund. He also worked as a summer intern for Corporate Research at Thomson Multimedia in 2000. He is a member of the Video and Image Processing Laboratory (VIPER Lab) at Purdue University, and his research interests include image and video compression, image feature detection, object recognition, and parallel algorithms.

He is a member of Tau Beta Pi, Eta Kappa Nu, Phi Kappa Phi, and the IEEE, was a 1993-1994 recipient of an Intel Foundation Graduate Fellowship, and is a reviewer for the IEEE.