A STUDY OF REAL-TIME AND RATE SCALABLE IMAGE AND VIDEO COMPRESSION

A Thesis
Submitted to the Faculty

of

Purdue University

by

Ke Shen

In Partial Fulfillment of the
Requirements for the Degree

of

Doctor of Philosophy

December 1997

To my parents, Lianqin and Eh,

for their love and encouragement.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my major advisor, Professor Edward J. Delp for his guidance and encouragement. His support has greatly contributed to my research and to the preparation of this thesis. I have truly benefited and enjoyed working with him.

I would also like to express my sincere appreciation to my graduate committee members, Professors Leah H. Jamieson, Zygmunt Pizlo and Ness B. Shroff, for their support.

I wish to thank the AT&T Foundation, the Rockwell Foundation, the Advanced Research Projects Agency, and the Concurrent Supercomputing Consortium at the California Institute of Technology for supporting this research.

I would also like to acknowledge my former colleague Dr. Mary L. Comer for planting the first seeds of my new wavelet based rate scalable video coding algorithm.

Finally, I would like to thank my officemates Eduardo Asbun, Gregory Cook, Sheng Liu and Paul Salama, not only for their support in preparing this thesis, but also for the memorable time we shared during the past several years.

DISCARD THIS PAGE

TABLE OF CONTENTS

Page

DISCARD THIS PAGE

LIST OF TABLES

DISCARD THIS PAGE

LIST OF FIGURES

Appendix
Figure Page

Appendix
Figure Page

Appendix
Figure Page

DISCARD THIS PAGE

ABSTRACT

Shen, Ke. Ph.D., Purdue University, December 1997. A Study of Real-time and Rate Scalable Image and Video Compression. Major Professor: Edward J. Delp.

In this thesis, we address two issues related to image and video compression. The first issue is on how to accelerate the speed of video compression using parallel processing, and the second is the development of a new approach to rate scalable coding of images and video.

1. Several parallel implementations of an MPEG1 video encoder on MIMD high performance systems, in particular the Intel Touchstone Delta and the Intel Paragon, are presented. In our approach, both *spatial* and *temporal* parallelism have been exploited. While the Paragon has the computation capacity to compress video sequences at a speed faster than real-time, we found that real-time performance cannot be achieved if the Input/Output (I/O) is not designed properly. We present several schemes, corresponding to different types of data parallelism, for managing the I/O operations and regulating the data flow. Using our algorithm, real-time MPEG compression of ITU-R 601 digital video sequences was achieved.

2. A new wavelet based rate scalable image and video compression algorithm is presented. A new embedded zerotree wavelet (EZW) approach for color image compression that exploits the interdependency between color components in the luminance/chrominance color space is described. The algorithm is known as *Color Embedded Zerotree Wavelet (CEZW)*. The new video compression algorithm uses motion compensation to reduce temporal redundancy. The prediction error frames and the intra-coded frames are encoded using *CEZW*. To address the error propagation problem inherit to rate scalable video compression, an *adaptive motion compensation (AMC)* scheme is designed. The rate scalable video compression algorithm is known

as *Scalable Adaptive Motion Compensation Wavelet (SAMCoW)*. We show that in addition to providing a wide range of rate scalability, *SAMCoW* achieves comparable performance to the more traditional hybrid video coders, such as MPEG1 and H.263. Furthermore, *SAMCoW* allows the data rate to be dynamically changed during decoding, which is very appealing for network oriented applications.

1. INTRODUCTION

Demand for video has been growing rapidly in the past few years, especially in areas such as education, communications, entertainment, and publishing [1]. Combining digital video, database technology, and communication networks technologies, the ability of delivering video, together with text, images and audio through networks upon requests makes video more accessible and more useful. Applications such as digital libraries, video databases (VDB), video-telephony/conferencing, and video-on-demand (VOD) are already in service or on the point of turning into practical services. However, capabilities such as storing huge amounts of data at a relatively low cost, efficiently organizing the data, retrieving, delivering, and presenting the requested data for easy access, need to be improved before these services can be broadly accepted.

One of the major obstacles to deploying digital video in many applications is the fact that the huge volume of uncompressed video data may easily overwhelm the available communication channels and storage systems. For example, a digital video sequence that conforms to the ITU-R 601 digital video recommendation (720x486 pixels per frame, 30 frames per second and 16 bits per pixel), which has a resolution comparable to the National Television System Committee (NTSC) analog video signal, has an uncompressed data rate of 168 megabits per second (Mb/s) [2, 3]. A typical two-hour movie would occupy approximately 150 giga-bytes of disk space. High definition television (HDTV) video sequences (1920x1080 pixels per frame, 30 frames per second and 16 bits per pixel) have an uncompressed data rate of approximately 1 gigabits per second (Gb/s) [4]. A two-hour movie in uncompressed HDTV format will occupy up to 900 gigabytes (GB) of storage space. However, a 6-MHz

NTSC transmission channel can accommodate a data rate of approximately 19 Mb/s using vestigial sideband (VSB) modulation [4]. By comparison, a dedicated DS-1 (or T1) line has a bandwidth of 1.5 Mb/s. The US Internet backbone, ANSNET, using DS-3 lines has a bandwidth of 45 Mb/s, while the future Internet backbone with optical based OC-3 lines will have a bandwidth of 155 Mb/s. None of the these channels has a bandwidth that is large enough to accommodate either uncompressed ITU-R 601 or HDTV signals. A 650 mega-bytes (MB) CD-ROM can store 5 seconds of uncompressed HDTV video or 31 seconds of uncompressed ITU-R 601 video. In videotelephony or videoconferencing, a QCIF (176x144 pixels per frames, 10 frames per second, and 12 bits per pixel) sequence has a uncompressed data rate of 3 Mb/s, which cannot be handled by a telephone line (POTS) with a bandwidth of 33.6 kilo-bits per second (Kb/s). One immediately sees the need for video compression. Many compression standards, such as ITU-T (CCITT) Recommendations H.261 and H.263, ISO standards MPEG1 and MPEG2 [5, 6, 7, 8, 9], have been proposed to address this problem.

The main objective of a video compression algorithm is to exploit both the spatial and temporal redundancy of a video sequence such that fewer bits can be used to represent a video sequence at an acceptable visual distortion. For different applications, different resolutions, visual quality and, consequently, different data rates are required. For HDTV signal, the target compressed data rate should be within the capacity of a 6-MHz NTSC transmission channel, or 19 Mb/s. For NTSC resolution (ITU-R 601) signals, a possible application is the digital versatile disk (DVD) which delivers up to 9.8 Mb/s and 17 GB of storage per disk (double-sided/dual layer). CIF (352x288) sequences are commonly used by video CD (VCD), popular in the European and Asian markets, which has a storage capacity of 650 MB per CD-ROM and data a rate of 1.5 Mb/s. Videotelephony and videoconferencing signals (QCIF sequences at 5–15 frames per second) often need to be transmitted on 33.6 Kb/s telephone lines.

In this thesis, we will address two issues related to video compression. One is how to accelerate the processing speed of video compression. The other is the scalable coding of digital images and video.

1.1 Parallel Processing in Video Compression

The time needed to compress a video sequence is an important issue. Real-time performance is necessary for many applications such as live digital television broadcasting and video conferencing. However, some applications may require "faster-than-real-time" performance. For example, suppose one wanted to convert an analog video library that contains 10,000 movie titles into a digital library. If each title is 2 hours in length, it would require more than 2 years to complete the project using real-time compression. "Faster-than-real-time" compression could reduce this process to several months.

The computation complexity of the algorithm, along with the compressed data rate, and the visual quality or distortion of the decoded video, are the three major factors used to evaluate a video compression algorithm. An ideal algorithm should have a low computation complexity, a low compressed data rate and a high visual quality (or low distortion) of the decoded video. However, these three factors usually cannot be achieved simultaneously. For an algorithm that generates compressed video with high quality and low data rate, the computation cost is usually high. It is highly likely that a compressed video sequence with a lower data rate will have a poorer visual quality after being decompressed. Among these three factors, the computation complexity directly affects the time needed to compress a video sequence. For example, the MPEG standard uses block based motion compensated prediction [10, 11, 12]. The MPEG encoding algorithms have to search for the motion vectors, which is a computationally intensive task. For example, if the sequence has a resolution of 720x480 pixels per frame (ITU-R 601), the search space is a 15×15 pixel array, and the motion vector search is performed on 20 frames (P frames and B frames) every

second, the processor must be able to handle

$$720 \times 480 \times 15 \times 15 \times 20 \approx 1.55 \times 10^9$$

pixels per second for the motion vector search [13, 14]. Although many fast algorithms have been proposed [10, 15, 16], the amount of computation is still overwhelming. Thus the MPEG algorithm is notoriously asymmetric, i.e. the amount of computation required for the encoding algorithm is much larger than that required for the decoding algorithm. Thus, real-time video compression is difficult to achieve and is very expensive especially when the image size is large, or high image quality and low data rates are required. The time needed to compress a video sequence is affected by the computation complexity of the algorithm, the efficiency in the implementation of the algorithm and the speed of the computation facilities. Parallel processing then becomes a natural choice to meet the large computation requirement for real-time video compression.

Video compression algorithms are usually suitable for parallel processing. A video sequence can be treated as a three dimensional signal, with two spatial dimensions and one temporal dimension. Many video compression algorithms are block based, which makes it feasible to parallelize an algorithm in the spatial domain, i.e. each processing element (PE) is required to process part of a video frame. Also a video sequence is a contiguous series of frames in time. Hence, it is also a nature choice to parallelize an algorithm in the temporal direction, i.e. each PE is assigned with one or more frames. Almost all hardware based video encoders incorporate some degree of parallel processing, including circuit level (on-chip) parallelism and system level parallelism. Many software based video encoders have also adopted parallel approaches, especially for networks of workstations (NOW) [17]. However, the speed of a NOW, under the bottleneck of network bandwidth and overall computation power, cannot fulfill the requirement of real-time video compression.

High performance parallel computers have been extensively used for scientific computing, such as solving partial differential equations (PDEs) and modeling the

molecular structure of DNA [18, 19, 20]. The tremendous computation power and the programmable nature of parallel computers make them very attractive for video server applications, especially video compression. In this thesis, we investigate the use of parallel computers for video compression. We present several schemes, corresponding to different types of data parallelism, for managing the I/O operations and regulating the data flow. We show their effect on increasing the overall speed of video compression. Using our algorithm, real-time MPEG compression of ITU-R 601 digital video sequences can be achieved.

1.2 Scalable Image and Video Compression

Many applications require that digital images/video be delivered over computer networks. The available bandwidth of most computer networks almost always poses a problem when video is to be delivered. A user may request an image or a video sequence with a specific distortion. However, the variety of requests and the diversity of the traffic on the network may make it difficult for an image or a video server to predict, at the time the video is encoded and stored on the server, the video quality and data rate it will provide to a particular user at a given time. One solution to this problem is to compress and store an image or a video sequence at different data rates. The server will then be able to deliver the requested image/video at the proper rate, given network loading and specific user request. This approach requires more resources to be used on the server in terms of disk space and management overhead. Therefore scalability, the capability of decoding a compressed sequence at different data rates, has become a very important issue in image and video coding. Scalable image/video coding has applications in browsing, digital libraries, video database system, video streaming, video telephony and multicast of television (including HDTV).

Scalability includes data rate scalability, spatial resolution scalability, temporal resolution scalability and computational scalability. The MPEG-2 video compression standard has incorporated several scalable modes, including signal-to-noise ratio (SNR) scalability, spatial scalability and temporal scalability [7, 21]. However, these

modes are layered instead of being continuously scalable. Continuous rate scalability provides the capability of arbitrarily selecting the data rate within the scalable range. It is very flexible and allows the video server to tightly couple the available network bandwidth and the data rate of the video being delivered.

A specific coding strategy known as *embedded rate scalable coding* is well suited for continuous rate scalable applications [22]. In embedded coding, all the compressed data is embedded in a single bit stream and can be decoded at different data rates. In image compression, this is very similar to progressive transmission. The decompression algorithm receives the compressed data from the beginning of the bit stream up to a point where a certain data rate requirement is achieved. A decompressed image at that data rate can then be reconstructed and the visual quality corresponding to this data rate can be achieved. Thus, to achieve best performance the bits that convey the most important information need to be embedded at the beginning of the compressed bit stream. For video compression, the situation can be more complicated since a video sequence contains multiple images. Instead of sending the beginning portion of the bit stream to the decoder, the sender needs to selectively provide the decoder with portions of the bit stream corresponding to different frames or sections of frames of the video sequence. These selected portions of the compressed data achieve the data rate requirement and can then be decoded by the decoder. This approach can be achieved if the position of the bits corresponding to each frame or each section of frames can be identified.

In this thesis we describe a new wavelet based coding algorithm for color images using a luminance/chrominance color space. Data rate scalability is achieved by using an embedded coding scheme, which is similar to Shapiro's embedded zerotree wavelet (EZW) algorithm [22]. The interdependence between color components are exploited in our algorithm. We denote this approach as the *Color Embedded Zerotree Wavelet (CEZW)* algorithm. Based on *CEZW*, we also propose a new continuous rate scalable hybrid video compression algorithm. We shall refer to this new technique as the *Scalable Adaptive Motion Compensated Wavelet (SAMCoW)* algorithm. *SAMCoW*

uses motion compensation to reduce temporal redundancy. The prediction error frames (PEFs) and the intra-coded frames (I frames) are encoded using *CEZW*. The novelty of this algorithm is that it uses an *Adaptive Motion Compensation (AMC)* scheme to eliminate quality decay even at low data rates. The nature of *SAMCoW* allows the decoding data rate to be dynamically changed so that the data rate can be adjusted to meet network loading.

1.3 Organization of the Thesis

In Chapter 2 an overview of the parallel implementation of image and video compression algorithms is presented. A temporal parallel algorithm for MPEG video compression, with real-time performance for CIF sequences, is described in Chapter 3. In Chapter 4, a spatial-temporal parallel MPEG compression algorithm that achieves real-time performance for ITU-R 601 sequences is presented. *CEZW* is described in Chapter 5. In Chapter 6, *SAMCoW* is described. A summary of the thesis and future work is given in Chapter 7. A postscript version of this thesis as well as the results presented in Chapter 5 for *CEZW*, including the original and decoded images, is available via anonymous ftp at
`ftp://skynet.enc.purdue.edu/pub/dist/delp/shen-thesis`.
The results of Chapter 6 for *SAMCoW* is available at
`ftp://skynet.ecn.purdue.edu/pub/dist/delp/samcow`.

2. PARALLEL IMPLEMENTATIONS OF IMAGE AND VIDEO COMPRESSION—AN OVERVIEW

The time needed to compress an image or a video sequence is very important in many applications. The computation complexity of the algorithm, along with the compressed data rate, and the visual quality of the decoded video, are the three major factors used to evaluate an image/video compression algorithm. An ideal algorithm should have a low compressed data rate, a high visual quality of the decoded image/video and a low computation complexity. Among these three factors, the computation complexity directly affects the time needed to compress a video sequence. Traditionally, the increase in execution speed for an algorithm has been obtained by increasing the processor speed. Unfortunately, many compression applications demand execution time that cannot be achieved using a single serial microprocessor. Parallel processing therefore becomes very attractive.

One of the characteristics of most image and video compression algorithms that make them feasible for parallel processing is the fact that the algorithms are block based. In block based compression methods, an image or a video frame is partitioned into non-overlapping spatial blocks which are coded separately. Furthermore, since a video sequence is a contiguous series of frames which represents a continuous action in time, it is natural to parallelize the algorithm in the temporal direction, i.e. each PE is assigned one or more frames.

Parallel video compression algorithms can be implemented using either hardware or software approaches [23]. In hardware approaches special parallel architectures can be designed to accelerate the computation [24, 25, 26]. Such an image or video compression device is feasible for applications such as a digital camera or real-time TV broadcasting. These devices are very expensive, e.g. a real-time MPEG2 encoder

for ITU-R 601 video costs in excess of $25,000. In many cases only a single video stream can be processed at a time. Software approaches include the use of parallel computers and networks of workstations [27, 28, 17, 29, 13, 14, 30, 31]. Usually a software implementation offers more flexibility, e.g. parameters can be adjusted easily and multiple video streams can be handled, which is essential for applications such as a video server or a video database. Portability may also be an advantage of a software implementation. A software approach is most suitable for algorithm development, digital library servers, or massive production of compressed digital video.

In this chapter we present an overview of techniques used to implement various image and video compression algorithms using parallel processing. The approaches used can largely be divided into three categories. The first is the use of special purpose architectures designed specifically for image and video compression. The use of VLSI techniques describes the second category. These include various chip sets for JPEG and MPEG. Software based approaches, or the implementation of algorithms on parallel or distributed computers, are the third category. Examples of this approach are the use of a massively parallel computer such as the MasPar MP-1, a coarse-grained machine such as the Intel Paragon, or a network of workstations (NOW).

2.1 DSP Arrays

One way to perform image and video compression in parallel is to use special architectures, usually an array of general-purpose digital signal processors (DSP). Examples of applications using a group of DSPs for image and video compression can be found in [32, 33]. In [32], 6 pairs of TMS320C30 DSPs and IMSA121 DCT chips are employed to implement the CCITT H.261 algorithm used in videotelephony. Each video frame is divided into 6 horizontal stripes, which are then assigned to each processing element (PE). To perform the motion compensation, blocks in the adjacent regions of the reference frame need to be read into each PE. For CIF (352×288) sequences, encoding and decoding can be performed at 15 frames per second when

motion estimation is not used. In [33], a parallel architecture using multiple DSPs is presented to implement the decompression algorithms for JPEG, H.261 and MPEG1 standards. In this architecture up to four floating point digital signal processors are connected to each other via fast serial links, while each processor has access to a globally shared memory.

In addition to the performance of the individual DSP chip, the architecture of the DSP array has a large impact on the execution speed of the algorithm. Okumura *et al.* [34] proposed a multistage switching network to balance the load of each DSP. It was found that with a balanced load the performance of the encoder is double that of a conventional configuration.

Recently, DSP chips have been developed specifically for video and image processing applications. These are known as video signal processors (VSP) or image digital signal processors (IDSP). These special purpose DSP chips usually incorporate on-chip parallelism into their architectures and are optimized to execute image processing tasks. VSPs with speeds up to 2 giga operations per second (GOPS) have been reported [35, 36, 37, 38, 39, 40, 41, 42, 43]. One example of VSPs is the Texas Instruments' Multimedia Video Processor (MVP), which is a single chip MIMD multiprocessor with crossbar shared memory [26]. On this chip, a reduced instruction set computer (RISC) is used to coordinate four DSPs. Real-time compression of CIF (320x240) video sequences has been reported using the MVP [44].

The use of DSP chips for image and video compression has the following advantages. First, the DSP chips are programmable and flexible, and can be configured for different image compression algorithms. Second, this method has relatively low cost. DSP chips and their peripherals, such as RAM and ROM memories, are quite inexpensive. The disadvantage of using general purpose DSP chips is the relatively slow arithmetic speed compared to an application specific VLSI design. This leads to the requirement of a large number of DSP chips, especially if the application requires real-time performance.

2.2 VLSI Approaches

Although DSP arrays have been used for implementing image and video compression algorithms, the success of these methods depend on the speed of the DSP. To overcome this problem, custom VLSI technology has been used to design application specific chip sets suitable for image and video compression. Custom VLSI chips usually utilize a RISC core, which has better support in operating systems and compiler design [24].

An example of this is Intel's i750 video processor, which is compatible with Intel's proprietary DVI motion video algorithm and the JPEG still image compression standard [45]. The i750 processor consists of two processor: a pixel processor and a display processor. The pixel processor performs decompression; the display processor performs post-processing operations such as YUV to RGB conversion. LSI Logic's L64702 JPEG coprocessor, with an 8.25 MB/s processing rate, can compress and decompress images with SIF resolution (352×240 pixels per frame) at 30 frames per second [46, 47].

C-Cube's CL950 MPEG video processor is a pipelined arithmetic unit that can decompress MPEG encoded SIF resolution video [48]. C-Cube Microsystems recently introduced the VideoRISC Compression Processor (VCP) [49, 25]. Eight VCPs are required to encode ITU-R 601 images using MPEG2 in real-time. Other examples of custom VLSI chips are C-Cube's JPEG and MPEG processors CL550, CL450 [48, 50]. VLSI chips have also been developed for DPCM encoders [51, 52]. Although the performance of these processors has greatly improved, none of these chips can encode real-time HDTV with a resolution of 1920×1080 pixels per frame.

Among the various methods that are employed to achieve high performance, on-chip parallelism is one of the most commonly used approaches [38, 39, 41, 53]. A P×64/MPEG encoder is reported in [54], and a video decoder for H.261/MPEG is

described in [55]. Both chips use a single instruction multiple data (SIMD) configuration. Some implementations have exploited the structure of the compression algorithm to make it more suitable for on chip parallel implementation [56, 57, 58, 59, 60].

2.3 Parallel and Distributed Computers

The use of arrays of DSPs or VLSI technology plays an important roll in increasing the processing speed of image/video compression. However, these approaches usually have specific architecture. A change in the algorithm or application often requires the redesign of the software as well as the architecture of the system. Flexibility is compromised. Parallel computers are very flexible in that different algorithms can be executed on the same computer without hardware reconfiguration. Almost every image/video compression algorithm can be implemented on a parallel computer. Parallel computers are also programmable in high level languages. A parallel computer is an ideal platform for the design of new parallel implementations of image and video compression algorithms. The speed and capacity of high performance parallel computers are very attractive to many applications where a large amount of imagery is involved. Examples include a multimedia on-demand video system and the processing of imagery used in remote sensing applications.

Implementations of MPEG-like encoders and DPCM on multiprocessor parallel systems have been described in [61, 62, 63]. Also Intel's DVI compression algorithm has been implemented on the MEiKO M40 system [64]. Another example of this approach is the implementation of Block Truncation Coding (BTC) on the multimicroprocessor PASM system [65, 66].

Recently, the JPEG still image compression algorithm has been implemented on the MasPar MP-1, a massively parallel SIMD computer [67, 68]. An image of size 1024×1024 was mapped to a 128×128 array of PEs so that each PE is assigned an 8×8 block of image data. Implementing the compression algorithm on a high performance parallel computer presented some unique problems. For example, it was shown that the most difficult part of the parallel implementation of the JPEG algorithm was not

in the implementation of the algorithm, but in the manner in which the compressed image data was output from the system [67, 68]. Also, the specific communication paths among the processors must be addressed. Although many operations, such as DCT and Huffman coding, can be performed within each processor, some operations such as motion estimation need to be performed on image data which is not local to the processor. The load balance between PEs may also be a problem [34]

In this thesis, we will present several parallel approaches to the implementation of the MPEG1 compression algorithm on MIMD, distributed memory super computers, in particular the Intel Touchstone Delta and the Intel Paragon. We will address the above problems and show that high performance parallel computers can be used for such applications.

3. PARALLEL MPEG COMPRESSION: THE TEMPORAL PARALLEL APPROACH

The Moving Pictures Experts Group (MPEG) has developed a standard, known as MPEG1, for the compression of digital video signals (and associated audio) at a data rate of 1.5 Mb/s [6, 11]. It utilizes motion compensated prediction to remove temporal redundancy and discrete cosine transform (DCT) coding to remove spatial redundancy. Motion compensated prediction is computationally intensive, however it provides a lower data rate than using just intraframe approaches for a fixed image quality. For a typical NTSC (ITU-R 601) video sequence motion compensated prediction can require as much as several gigaflops per second. This immediately indicates that MPEG will require some effort at real-time implementation.

Several software approaches have been described for the implementation of MPEG1. A software MPEG1 encoder has been developed at the University of California at Berkeley [17] that can compress video at a rate of 1.2 frames per second for images with spatial resolution of 352x288 (CIF) on a Sun SPARCstation 10. To increase the execution speed, this encoder is able to distribute the compression task to a group or a cluster of workstations interconnected via a LAN or the Internet. For example, on 6 Sun workstations connected by a 10 Mb/s Ethernet a typical execution rate of 4.7 frames per second can be obtained. A software-based parallel MPEG2 video encoder has also been reported [30]. However neither of these software encoders have reported real-time performance (30 frames per second).

A video sequence can be viewed as a 3-dimensional (3-D) signal with two dimensions in the spatial domain and one in the temporal domain. Since many video compression algorithms are frame/block based, video data can be distributed to the

processors frame-wise (*temporal parallelism*) or block-wise (*spatial parallelism*) without changing the overall computation complexity of the algorithm significantly. In spatial parallelism [27, 28, 29, 14, 30, 69], each frame of a video sequence is divided into several parts containing an integer number of blocks of pixels and each part is encoded by a processing element (PE). The entire video sequence is processed in a frame-by-frame fashion, which results in a minimum delay. However, there is an upper limit on the number of PEs that can be used because of the limited spatial resolution of the video sequence [27, 69]. Also a massive spatial parallel algorithm usually needs to tolerate a relatively high communication overhead. In [27], strict spatial parallelism was used and real-time performance on the compression of CIF (352×288 pixels per luminance frame) video sequences was achieved. A maximum compression speed of 31.14 frames per second was obtained using 330 processors on an Intel Paragon. However, in their computation of compression speed (execution time), the time needed for the I/O operations was not taken into account. When I/O operations are included, the algorithm will not obtain real-time performance. In temporal parallelism, different PEs are assigned to process different video frames of a video sequence [17, 13, 31]. There is no upper limit on how many PEs can be used since a typical 2 hour motion picture contains more than 200,000 frames. A problem with this approach is that even though the overall performance (throughput) can be faster than real-time the compression of a single frame may be slower than real-time, which results in a constant delay.

In this chapter we describe a temporal parallel implementation of an MPEG1 encoder on the Intel Touchstone Delta and Intel Paragon [70, 71]. Due to the similarity between MPEG1 and MPEG2, our implementation and results can be extended to MPEG2 (main profile, main level).

3.1 An Overview of MPEG1

The MPEG1 standard defines the syntax of the bit stream of a compressed digital video sequence [6, 11]. In other words, the decoder is defined and the implementation

of the encoder is open to individual designs. MPEG divides the compressed video frames into three types: I pictures, P pictures and B pictures (Figure 3.1). I pictures, the *intracoded pictures*, are encoded without reference to other frames, exploiting only the spatial correlation within a frame. P pictures, the *predictive coded pictures*, are encoded using motion compensated prediction from a past I picture or P picture. The *bidirectionally-predictive coded picture*s, or B pictures, require both past and future reference frames (I pictures or P pictures) for motion compensation.

An MPEG1 encoder needs to specify which frames of the input video sequence are to be compressed as an I, P, or B picture. This pattern of I, P and B pictures may (or may not) be repeated throughout the the video sequence. An example of a pattern of frames is IBBPBBPBBIBBP$\cdots$, where I, P, and B indicate an I picture, a P picture and a B picture, respectively. Hence an input sequence could be:

$$\mathrm{I_1B_2B_3P_4B_5B_6P_7B_8B_9I_{10}B_{11}B_{12}P_{13}B_{14}B_{15}P_{16}B_{17}B_{18}I_{19}B_{20}B_{21}P_{22}B_{23}B_{24}P_{25}B_{26}B_{27}I_{28}\cdots}, \tag{3.1}$$

where the subscripts indicate the sequence frame numbers. The order of the frames in the compressed sequence is rearranged according to the order in which an MPEG decoder can decompress the frames with minimum frame buffering (a maximum of three frame buffers). Hence, at the output of the encoder for the above sequence, the frame order has the form

$$\mathrm{I_1P_4B_2B_3P_7B_5B_6I_{10}B_8B_9P_{13}B_{11}B_{12}P_{16}B_{14}B_{15}I_{19}B_{17}B_{18}P_{22}B_{20}B_{21}P_{25}B_{23}B_{24}I_{28}B_{26}B_{27}\cdots}. \tag{3.2}$$

In an MPEG encoded sequence, several consecutive frames are combined to form a structure known as a group of pictures (GOP). While a GOP can contain one or more I pictures, the first picture in a GOP must be an I picture. The existence of GOPs facilitates the implementation of features such as random access, fast forward, or fast and normal reverse playback [11].

The image color space used in the MPEG1 is the YC_rC_b space, in which Y represents the luminance component and C_r and C_b represent the chrominance components, or the color difference components [6, 72, 8]. For an original video frame, each spatial position (pixel) is represented by a Y component, a C_r component and a C_b component. Thus, a video frame consists of 3 images of the same size, each of which corresponds to the Y, the C_r or the C_b component. This is known as the 4:4:4 format [8, 72]. Observations have shown that the human visual system (HVS) is less sensitive to chrominance signals than to luminance signals. In MPEG1, the chrominance components are subsampled with respect to the luminance component by half in both vertical and horizontal directions, known as the 4:2:0 format. A 2:1 compression is immediately achieved.

A video frame is divided into non-overlapping blocks of pixels, known as macro-blocks, each of which contains 16x16 pixels of the luminance component, 8x8 pixels of the C_r chrominance component and 8x8 pixels of the C_b chrominance component. An 8×8 luminance or chrominance pixel array is known as a block. Hence a macro-block contains 4 luminance blocks and 2 chrominance blocks (one C_r block and one C_b block). A video frame can also be divided into slices, each of which consists a series of an arbitrary number of macro-blocks in raster-scan order from left to right and top to bottom in the frame. Slices in a frame are non-overlapping and covering all the macro-blocks.

In the compression of I pictures, discrete cosine transform (DCT) is used on each block to decorrelated the pixel values. The DCT coefficients are quantized. Different quantizer step sizes, set by the quantizer scale, can be used to control the data rate. An increase in the quantizer step size results in a coarse quantization and hence, a decrease in the output data rate. Differential pulse coding modulation (DPCM), run length coding and variable length coding (VLC) are then used to code the quantized coefficients. In the compression of P pictures and B pictures, motion compensated prediction is used. Motion associated to a macro-block in a P or B picture is represented by a two-dimensional vector, known as a motion vector, which

specifies where to retrieve the macro-block from the reference frame (Figure 3.2). The motion vectors are then encoded using DPCM and VLC. A predicted picture is formed by replicating the pixels from the reference frame at new locations according to the motion vectors. The difference picture, or the error picture, taken between the current picture and the predicted picture is encoded using the same DCT-based techniques used for the I pictures. If a motion vector cannot be found according to the matching criterion, the macro-block will be coded as an intra-coded macro-block (similar to the macro-blocks in an I picture).

The generation of the motion vector fields (MVFs) is up to individual algorithm design. The motion vector associated with a macro-block is usually obtained by matching the macro-block with pixel arrays within the spatial search range in the reference frame. The displacement between the macro-block and the best matched pixel array in the reference frame is used as the motion vector. The choice of a matching criterion is open to individual design. Many different strategies can be used to search for motion vectors, including full search and logarithmic search [10, 12]. If full search, or exhaustive search, is used, all macro-block sized pixel arrays within the chosen spatial search range need to be evaluated, which makes the computation very intensive. Other methods, such as logarithmic search [10], use strategies to control the search process so that only a subset of all the possible displacements are evaluated and the amount of computation is reduced. For example, in logarithmic search (Figure 3.3) a grid of 9 pixel displacements is evaluated and the search continues based on a smaller grid centered on the position of the best match. For the macro-block centered at 1', pixel arrays centered at all the pixels marked 1 are examined. If the pixel array at 1* has the best match, the pixel arrays centered at 2 will be examined. Eventually the pixel arrays centered at 3 will be examined and the one with the best match is chosen. In the example, the motion vector will be a vector from pixel 3* to 1'.

Since I and P pictures are used as reference pictures, it is necessary to maintain a relatively high video quality for these pictures. On the contrary B pictures which are never used as reference pictures can tolerate a relatively poorer video quality. Thus

the quantization steps for I and P pictures are set to be smaller, which correspond to higher compressed data rates, than those for B pictures. Also, since the motion compensated prediction is used for P and B pictures, the data needed to be coded in P and B pictures have lower energy, which correspond to lower compressed data rates, than I pictures. Therefore, usually I pictures have the highest data rate and the lowest motion artifacts, while B pictures have the lowest data rate and the highest motion artifacts. The typical data rate of an I picture is 1 bit per pixel while that of a P picture is 0.5 bits per pixel and for a B picture, 0.25 bits per pixel. Obviously the use of P pictures and B pictures yields a much lower data rate. However, it takes much longer time to compress a P or B picture because the encoding algorithm need to search for the motion vectors.

It it worth of knowing that the main profile main level MPEG2 has a very similar structure to the MPEG1 algorithm described in this section.

3.2 An Overview of the Intel Paragon and the Intel Touchstone Delta

While our approach is very general and could be extended to any MIMD distributed memory architecture, in our experiments we used the Intel Touchstone Delta and Intel Paragon as the implementation platform. The Intel Touchstone Delta consists of 512 (16x32) Numerical Nodes (Figure 3.4) [71]. Each node is an Intel i860 processor which operates at 40 MHz with 16 MB of memory and has a peak speed of 60 Mflops double-precision or 80 Mflops single-precision. The overall peak speed of the Touchstone Delta is 32 Gflops. Each Numerical Node is attached to one of the mesh routing chips (MRCs) which are interconnected via a network with a two-dimensional mesh topology. Since these nodes do not share memory, they must exchange messages through the network to share information. The mesh is connected to a Concurrent File System (CFS) with 90 Gbytes of disk space. The CFS consists of 64 disks which are served by 32 I/O nodes, each of which serves two disks. The file system distributes file blocks to all available disks using algorithms for reading and writing that allow

several PEs to use the concurrent disks simultaneously [71]. The Touchstone Delta is located at the Concurrent Supercomputing Consortium at Caltech.

The Intel Paragon XP/S parallel computer is a typical distributed memory, multiple instruction streams multiple data streams (MIMD) parallel computer [70]. It has a similar architecture to that of the Touchstone Delta and uses i860 XP processors which operate at 50 MHz and have peak floating point performance of 75 Mflops double-precision or 100 Mflops single-precision (Figure 3.5).

The mesh is connected to the Parallel File System (PFS), which distributes file blocks to all available disks using algorithms for reading and writing that allow several PEs to use the file systems simultaneously. The number of numeric nodes and the number and size of PFSs in a Paragon computer vary relative to each configuration.

The Paragon computers we used for our experiments are the system located at the Concurrent Supercomputing Consortium at Caltech which has 512 Numerical Nodes and the one located at Purdue University which has 140 Numerical Nodes. The total computation capacities exceed 38 Gflops and 10 Gflops, respectively. Both systems have multiple PFSs.

The Touchstone Delta and Paragon have been used extensively in study of problems in scientific computing [18, 20].

3.3 Parallel Implementation

The parallel algorithms we developed use the Single Program Multiple Data (SPMD) model of parallelism [73]. The data stream is divided in temporal parallel and distributed to different PEs. We started by directly mapping the Berkeley Encoder to the Touchstone Delta and Paragon. The algorithm was then modified and a new I/O algorithm was developed.

3.3.1 Mapping of the Berkeley Encoder

The Berkeley Encoder is able to run on several interconnected workstations by distributing the video data via a network file system (NFS) and passing messages

through TCP/IP sockets [17]. Figure 3.6 shows the processing architecture used when encoding a video sequence in parallel. The Master Server passes messages to the Slaves via TCP/IP sockets directing the Slaves to compress selected frames. The video sequence is stored on an NFS, with each video frame stored as an individual file. The Slaves read the appropriate frames from the NFS and compress the frames independently. The compressed frames are written to the NFS as individual files. The Combine Server reads the compressed frames from the NFS in the correct order, adds a sequence header, GOP headers and a sequence ender when appropriate, and outputs the compressed data as a complete MPEG sequence in a single file.

Direct mapping of the Berkeley Encoder to the Touchstone Delta and Paragon is straightforward if each PE is treated as a workstation. We divide the *N* processing elements (PEs) into one Control PE and *N-1* Compression PEs. The Control PE acts similar to the Master Server in the Berkeley encoder and the Compression PEs act similar to the Slaves. The communication between the PEs is performed by message passing through the mesh network. The video sequence to be compressed is stored on the CFSs of the Touchstone Delta or PFSs of the Paragon.

Because the compressed video data has different length for each frame and must be output in the correct order, it is not feasible to output a single MPEG sequence in parallel from different compression PEs. To write the compressed data into a single file, a PE would have to wait for other PEs to complete their output operations, which virtually makes the algorithm sequential. In our implementation, for the sake of parallelism, the function of the Combine Server in the Berkeley encoder is eliminated. Instead a list of file names of the compressed data is generated by the Control PE. The only task that is not fulfilled in our program is the concatenation of the individual files into a fully MPEG1 compatible sequence.

The direct mapping of the Berkeley Encoder does not execute fast enough. The algorithm obtained from direct mapping of the Berkeley Encoder to the Touchstone Delta and Paragon does not execute fast enough. The overall execution speed on the Touchstone Delta in terms of frames per second, shown in Figure 3.7(a), is obtained by

dividing the number of compressed frames by the overall execution time. Obviously, it runs slower than the real-time rate of 30 frames per second. By examining the running times of different modules of the software, we found that the average I/O access time accounts for more than half of the total execution time. Excluding the I/O time, the execution speed is linear with respect to the number of PEs and can exceed 500 frames per second (Figure 3.7(b)). Also, we found that the speed decreases when the number of PEs exceeds 100, which we believe is also caused by I/O contention. To further examine this effect, we observed the amount of time needed to read 512 CIF images as a function of the number of PEs. The images were equally distributed to each PE and the average time used by the PEs is shown in Figure 3.8.

If the parallel computer had an ideal I/O system, one would expect the average time to decrease linearly as the number of PEs increases. Our results indicate that the average time decreases linearly only if the number of PEs is less than 32. Furthermore, there is a decrease in the I/O speed when the number of PEs exceeds 128.

The approach used in the Berkeley Encoder relative to the I/O operation is not suitable for parallel computers such as the Touchstone Delta and the Paragon. In the CFS or PFS, I/O nodes and the mesh networks of the Touchstone Delta and the Paragon are optimized for operations such as opening a file (parallel mode), reading/writing a large amount of data from/to the file and then closing the file [71, 70]. In the Berkeley Encoder each file contains a single original frame at the input and a single compressed frame at the output. Hence, a large number of files have to be opened, read/written or closed simultaneously. This causes extreme I/O contention on the Touchstone Delta and Paragon and is very inefficient. To overcome this bottleneck we developed a new I/O algorithm.

3.3.2 A New I/O Algorithm

As mentioned above, the Touchstone Delta has 64 disk systems which are served by 32 I/O nodes. The I/O subsystem uses an algorithm to distribute file blocks on multiple disks so that data from a single file can be transferred between the disks

and the mesh network in parallel. This feature makes reading or writing of a large file very efficient. However, it is not optimal for our application. We need to open a large number of files at the same time from different PEs. Since opening a large number of small files for input and output is very inefficient for the Touchstone Delta or Paragon, we grouped sections of consecutive frames into single files. It is ideal if a PE can open and read from a single file that contains all the frames assigned to it, including the necessary reference frames. A similar argument holds for the output operation. Since an MPEG sequence requires the output frames to be in a specific order, an arbitrary number of consecutive frames cannot be written into a single file after compression. For example, in the sequence we described in (1) if the section $I_1B_2B_3$ is assigned to a PE, the compressed data cannot be written in a single file because frame P_4 needs to be inserted between I_1 and B_2B_3. Hence, there are only a limited number of ways to break a video sequence into sections. For example the sequence described in (1) could be grouped as $I_1B_2B_3P_4B_5B_6P_7$, $P_7B_8B_9I_{10}B_{11}B_{12}P_{13}B_{14}B_{15}P_{16}$, and $P_{16}B_{17}B_{18}I_{19}B_{20}B_{21}P_{22}B_{23}B_{24}P_{25}$ in three files, and then assigned to three PEs, respectively. The first PE will compress the frames $I_1 \cdots P_7$. The second and third PEs will compress the frames $B_8 \cdots P_{16}$ and $B_{17} \cdots P_{25}$, using P_7 and P_{16} as reference frames, respectively. The output will be in three files $I_1P_4B_2B_3P_7B_5B_6$, $I_{10}B_8B_9P_{13}B_{11}B_{12}P_{16}B_{14}B_{15}$, and $I_{19}B_{17}B_{18}P_{22}B_{20}B_{21}P_{25}B_{23}B_{24}$, each of which is a GOP. The only task left is to concatenate these three files to obtain an MPEG sequence.

Often there are more PEs executing I/O operations than the number of I/O nodes. Hence, each I/O node has to serve several PEs at the same time, which increases the contention problem. We want each I/O node to serve only one PE at a time. The Touchstone Delta has a system call, **restrictvol()**, that restricts the disk volumes to which a file can be allocated. By pre-allocating each file to a single disk volume, we can guarantee that each file is served by a single I/O node. In practice, files are allocated to disk volumes in a round-robin fashion, e.g. File1 on Volume1, File2 on Volume2, ..., File64 on Volume64, File65 on Volume1, etc. Since there are only 32

I/O nodes, 32 PEs can be served at the same time. Other PEs that need to perform I/O operations have to wait in a queue which serves I/O requests on a first-come-first-serve basis. In this way, the entire compression task is pipelined in the following sense. Certain number of PEs can perform I/O operations at the same time while other PEs that need to perform I/O operations must wait in a queue. When a PE finishes its I/O operation and starts computation, one I/O node is freed such that a PE waiting in the queue can be served. Since the amount of time used to output compressed data is much less than that used to input the original video data, this I/O queue is used for input only.

On the Paragon, the feature of restricting the disk volumes to which a file can be allocated is not available. However the I/O queue is still used to limit the number of PEs performing I/O operations simultaneously.

A block diagram of the data flow in the temporal parallel algorithm is shown in Figure 3.9.

3.4 RESULTS

The parallel algorithms have been tested on different video sequences, including CIF (360x288) versions of the *Salesman* and *Flowergarden* sequences and a ITU-R 601 (704x480) version of the *Football* sequence. The video source is stored in the YUV color space, which can be transformed to the YC_rC_b color space, with the chrominance components, U and V, subsampled to 4:2:0 format [72]. In our experiments, we set the quantizer scales to be 8 for I pictures, 10 for P pictures and 25 for B pictures and the frame pattern to be IBBPBB. The GOP size is set to 12 frames. The logarithmic search scheme was used on half pixel displacements with the search range set to 10 pixels. The reduction in data rate after compression is significant. For example, the *Football* sequence has an uncompressed data rate of 120 Mb/s, while the average compressed data rate is reduced to 3.8 Mb/s, which yields a compression ratio of 31:1. The two CIF sequences have an uncompressed data rate of 38 Mb/s, while the average compressed data rate is 1.5 Mb/s for the *Flowergarden* sequence and 400

Kb/s for the *Salesman* sequence, which makes the compression ratio to be 24:1 and 92:1, respectively. While the performance of the algorithm in terms of data rate and image quality is satisfactory, the execution time becomes the major concern. The results we report in this paper are based on the *Salesman* sequence which consists of 450 frames. We expanded this video sequence to more than 9000 frames (5 minutes) by replicating the sequence.

From the data shown above, one can see that I/O contention is the bottleneck in our encoder. Our new I/O algorithm improves the I/O efficiency and hence the overall execution time. The execution speed of the encoder on the Touchstone Delta with the modified I/O algorithm is shown in Figure 3.10. An execution speed of more than 41 frames per second has been obtained on 144 PEs. Notice that the I/O queue algorithm speeds up the compression by almost 100%. The comparison of the execution times of the algorithms with and without the I/O queue on the Paragon is shown in Figure 3.11. Although the function of restricting the disk volumes to which a file can be allocated is not available on the Paragon, limiting the number of PEs allowed to perform I/O operations simultaneously reduces the I/O contention efficiently. A speed of more than 41 frames per second has been obtained on 100 PEs. Hence faster than real-time compression has been achieved.

For ITU-R 601 video sequences, the overall performance of the temporal only parallel algorithm on the Paragon in terms of compression speed is shown in Figure 3.12. We can see that the algorithm has near linear speedup when the number of processors involved are less than 380. When the number of PEs exceeds 380, increasing the number of PEs does not result in any more speedup. The reason is that the I/O utilization is so close to the system's I/O bandwidth limit that the increase in the number of processors only increases I/O contention. This effect can be seen from Figure 3.13 and Figure 3.14. In Figure 3.13, the percentage of time used by a certain module is defined as the ratio between the summation of the time used for this module on all the processors and the summation of the total running time on all the processors. In Figure 3.14, the virtual number of processors used for a certain module is defined as

the percentage of running time used for this module multiplied by the total number of processors. This number indicates how many processors are required for a certain module if 100% of the running time of these processors are devoted to this module. We can see that while the number of PEs increases, the percentage of the computer cycles used on compression decreases and the percentage of computer cycles used on I/O increases dramatically (Figure 3.13). The overall computation power that is used on compression does not increase when the number of PEs exceeds 380 (Figure 3.14).

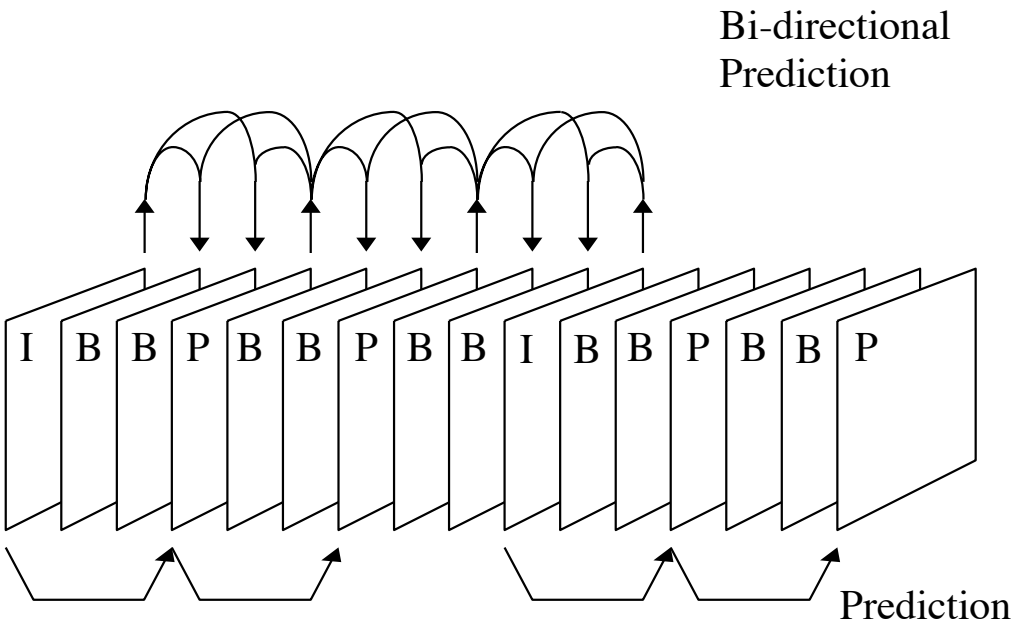


Fig. 3.1. An example of a video sequence with the I picture, P picture or B picture assigned to each frame.

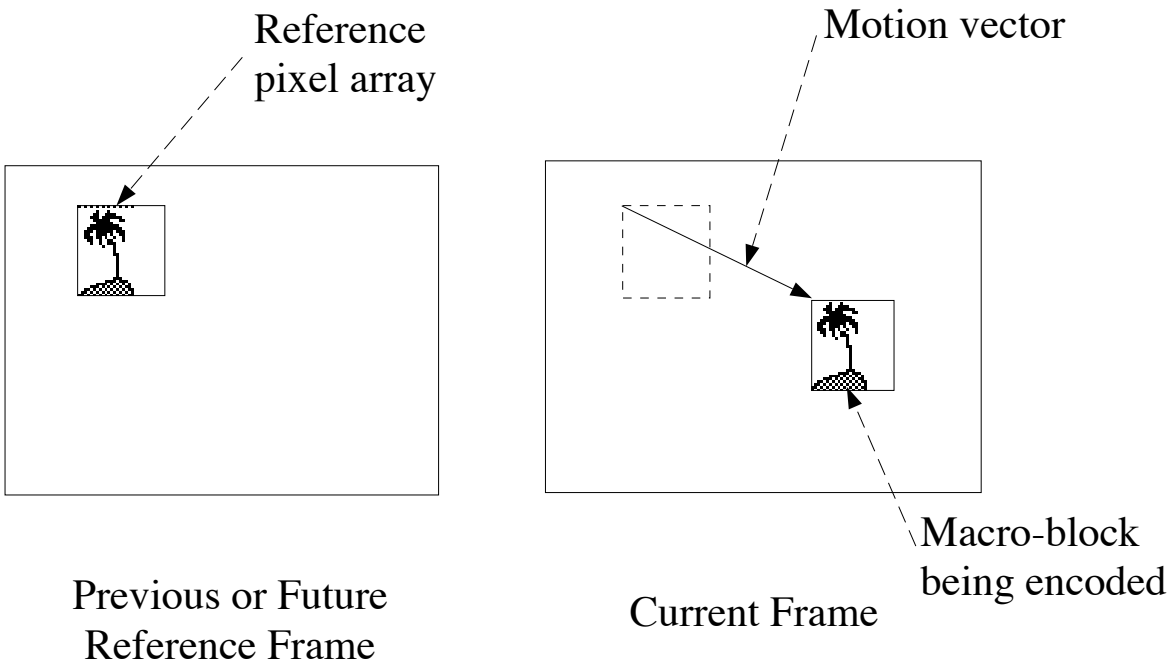


Fig. 3.2. Motion is described by a two-dimensional vector that specifies where to retrieve a macro-block from the reference frames.

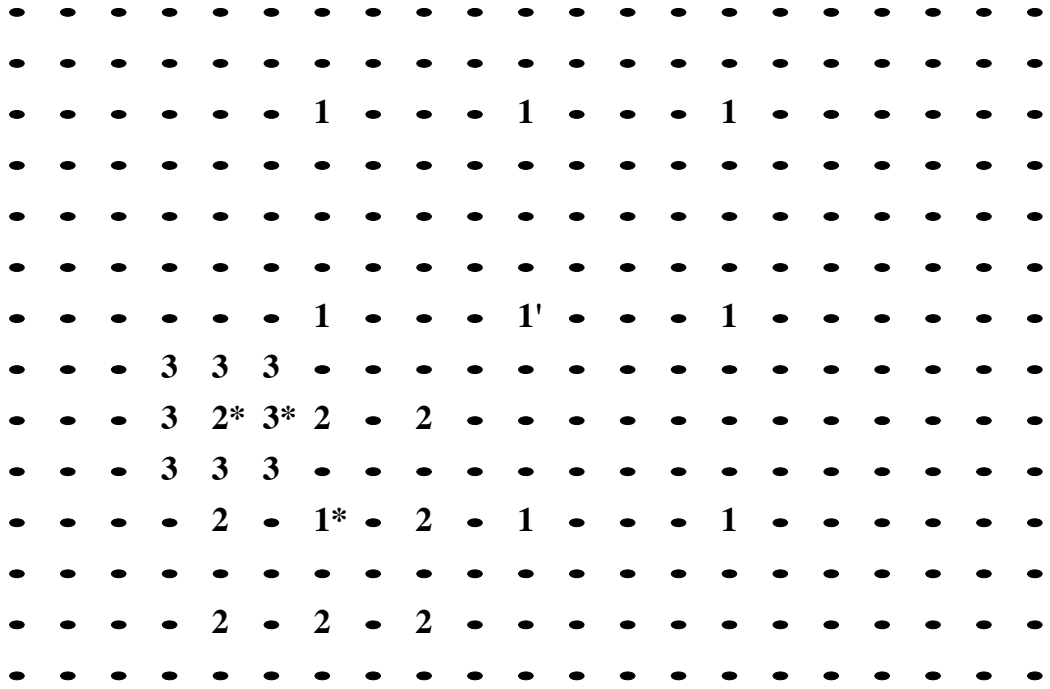


Fig. 3.3. An example of logarithmic search method using integer pixel displacements

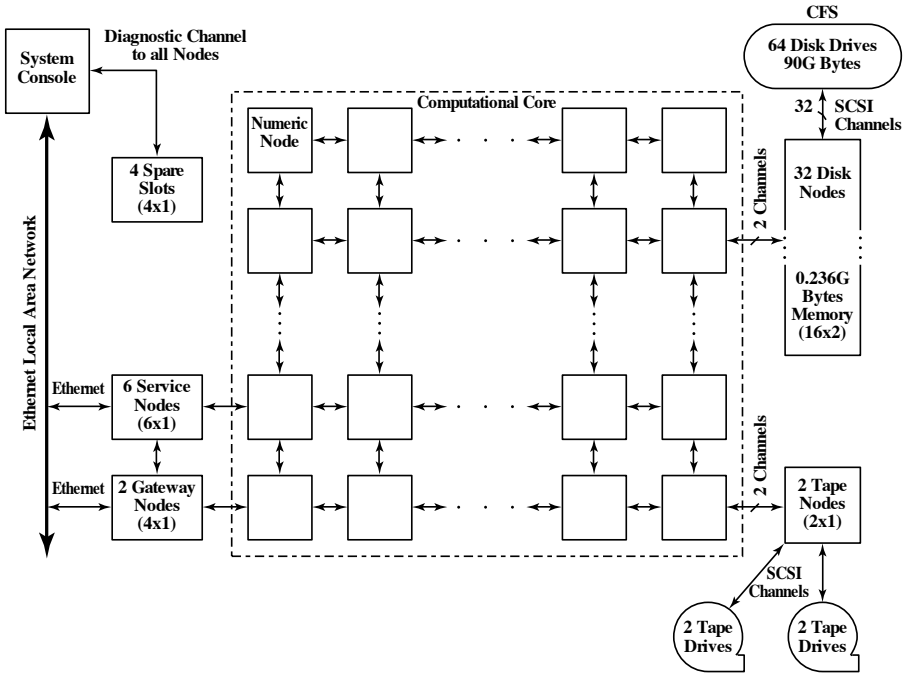


Fig. 3.4. The Intel Touchstone Delta system architecture.

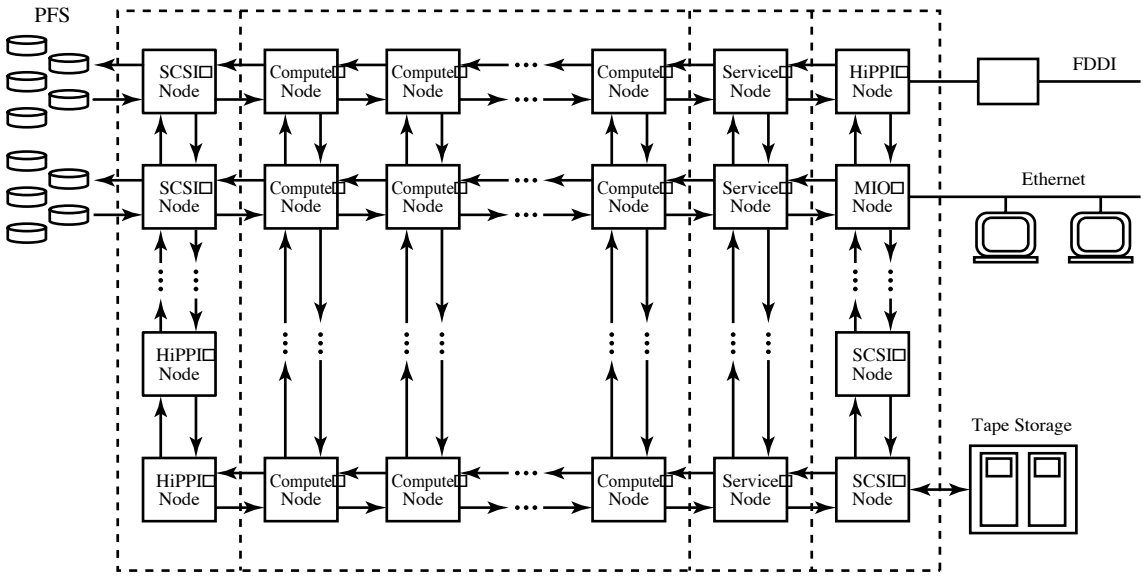


Fig. 3.5. The Intel Paragon XP/S system architecture.

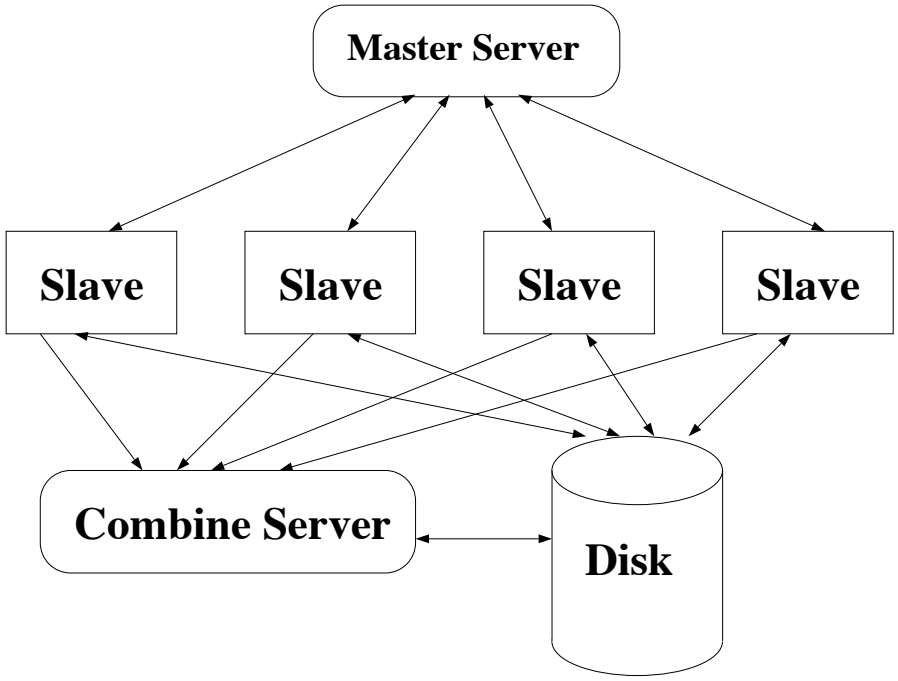
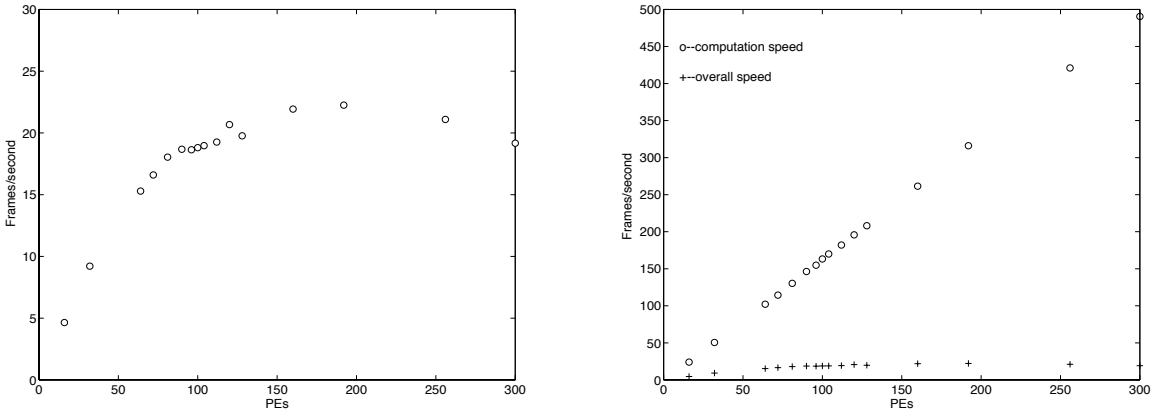


Fig. 3.6. Diagram of the Berkeley Encoder.



(a) (b)

Fig. 3.7. (a) The overall performance on the Touchstone Delta of the parallel algorithm obtained by directly mapping the Berkeley Encoder. (b) The performance obtained by examining only the computation time. The overall performance is also shown for comparison.

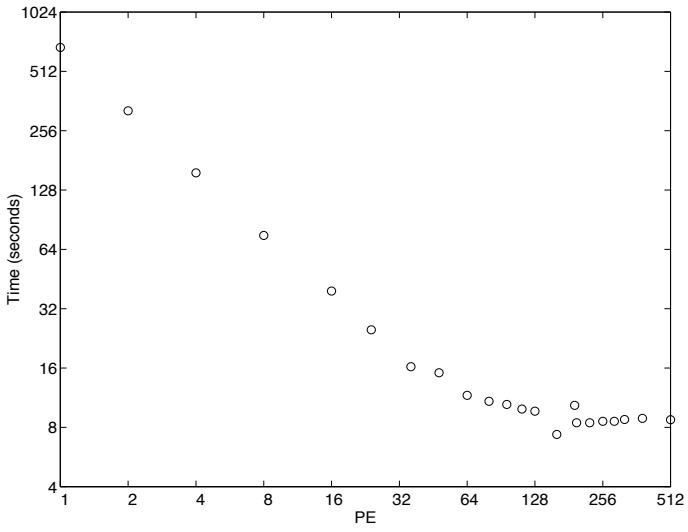


Fig. 3.8. Time needed to read a total of 512 images on the Touchstone Delta. The numbers of images are equally assigned to each node.

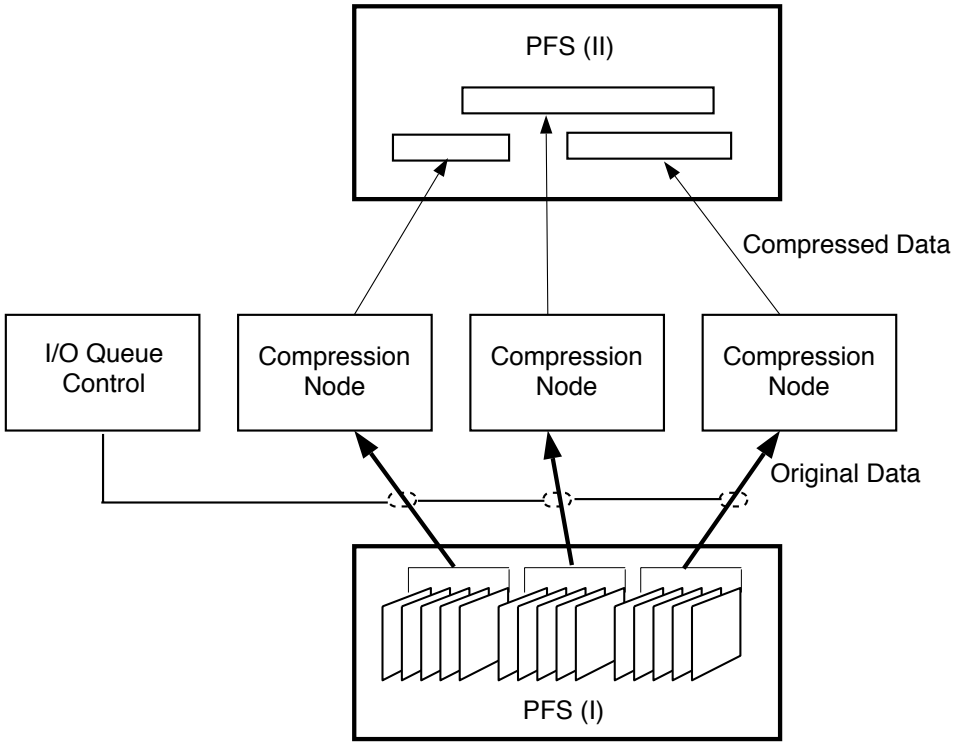


Fig. 3.9. Block diagram of the temporal algorithm.

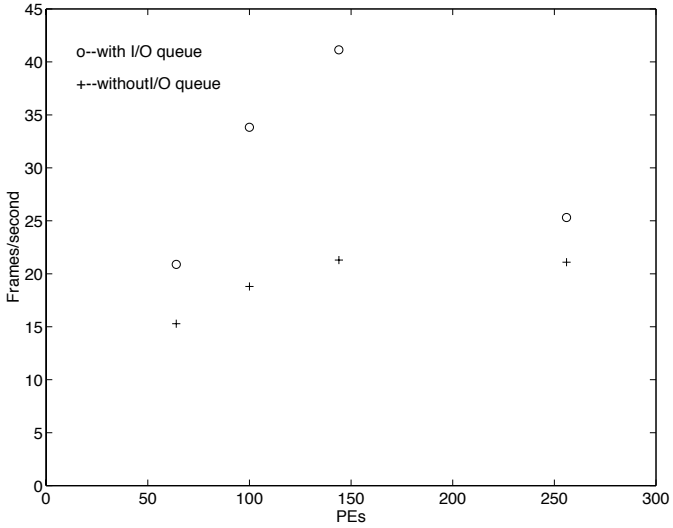


Fig. 3.10. The overall performance on the Touchstone Delta with the new I/O algorithm. The performance before modification is also shown for comparison.

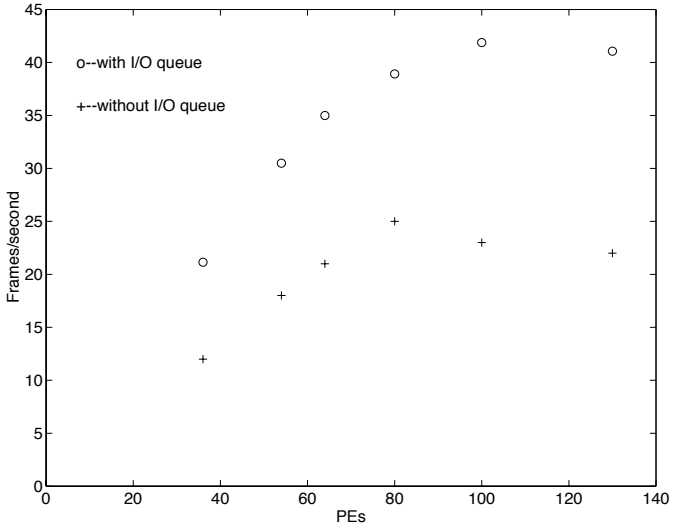


Fig. 3.11. The overall performance on the Paragon with the new I/O algorithm. The performance of the algorithm without the I/O queue is also shown for comparison

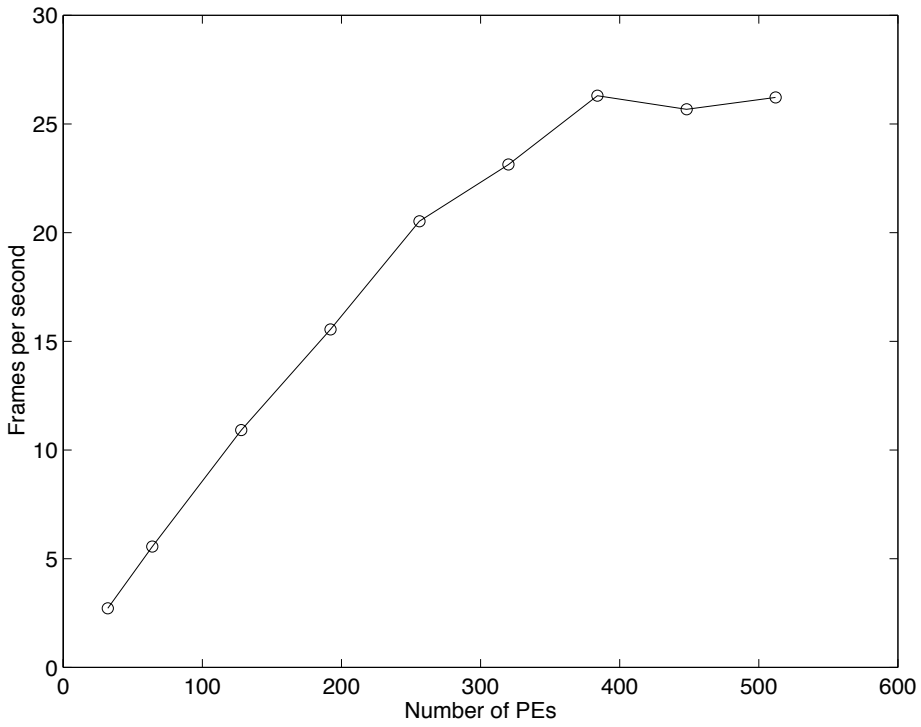


Fig. 3.12. The overall speed on the Paragon for the compression of ITU-R 601 video using temporal parallelism.

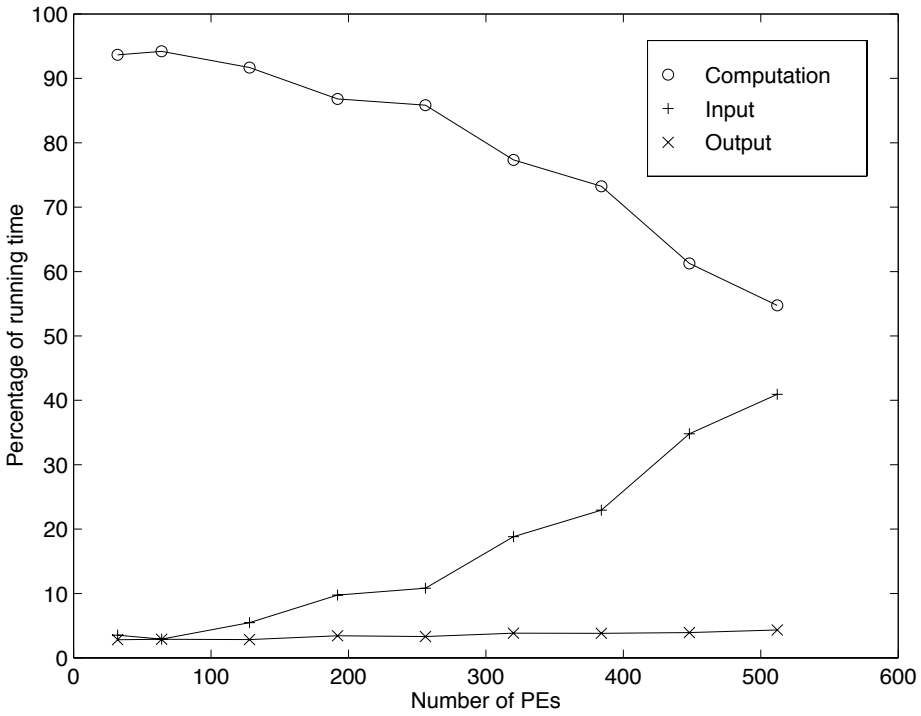


Fig. 3.13. The percentage of running time on the Paragon used by different modules in the temporal parallel algorithm.

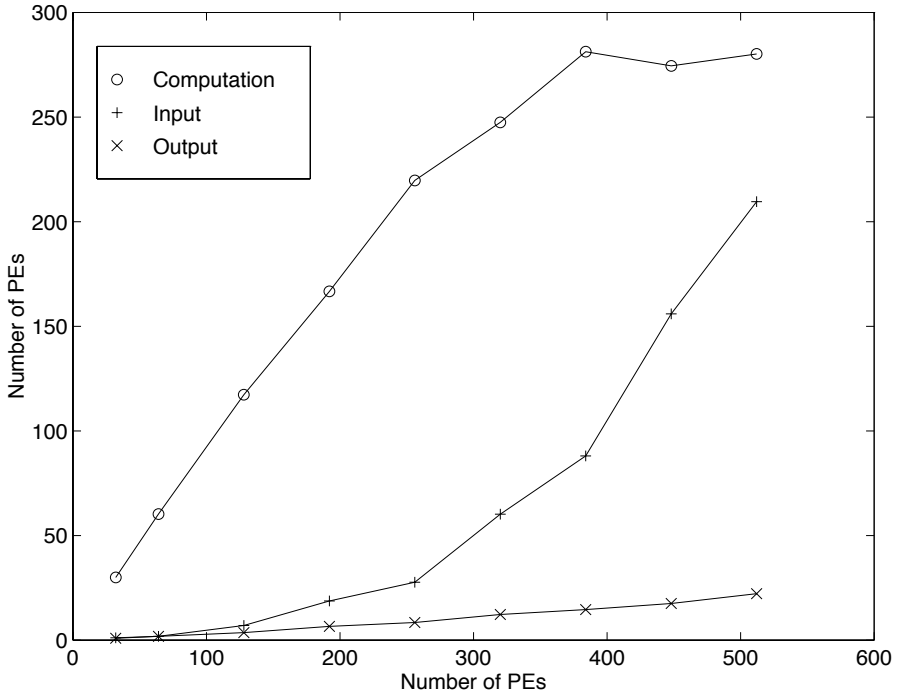


Fig. 3.14. The virtual number of processors on the Paragon used for different modules in the temporal parallel algorithm

4. PARALLEL MPEG COMPRESSION: THE SPATIAL-TEMPORAL PARALLEL APPROACH

The main objective of the work has been to develop parallel implementations that provide real-time throughput. In our experiments, we found that a single node on the Paragon can compress a ITU-R 601 video sequence at a speed of 0.09 frames per second with 95.4% of the time used for computation. Since the parallel algorithm does not reduce the overall amount of computation required, to achieve real-time performance 320 nodes are needed if 100% of the time is used for computation. Thus it is critical to minimize the time spent on tasks other than computation, such as I/O, communications and data distribution/assemblage, in the implementation. For a high performance parallel computer, such as the Intel Paragon, which is optimized for scientific computation, the problem can be difficult when a huge data throughput needs to be maintained. Recent work has shown that the performance of the I/O system can greatly affect the performance of the whole algorithm in parallel image/video processing where the data set is huge [27, 28, 13]. The efficiency of the parallel algorithms is determined by how the algorithm can reduce the overhead, including the I/O operations, communications and data distribution/assemblage. For the 512-node Paragon used in our experiments, the goal of our work can be reiterated as follows: develop a parallel algorithm that, on average, less than $1 - 320/512 = 37.5\%$ of the time is used for I/O, communications and data distribution/assemblage.

In [27], strict spatial parallelism was used and real-time performance on the compression of CIF (352×288 pixels per luminance frame) video sequences was achieved. A maximum compression speed of 31.14 frames per second was obtained using 330 processors on an Intel Paragon. In the previous chapter, we presented a temporal parallel approach which can provide faster than real time compression for CIF video

sequences using the Intel Touchstone Delta and the Intel Paragon. However, for ITU-R 601 images (720×486 pixels per luminance frame) the compression speed is less than 30 frames per second using the temporal parallel algorithm. The reason is that a ITU-R 601 frame is almost 4 times the size of a CIF sized frame, which increases both the computation and the I/O time.

In this chapter, we will introduce the spatial-temporal parallel approach implemented on the Intel Paragon which will further reduce the I/O contention. We will show that real-time performance can be achieved for ITU-R 601 video.

4.1 Spatial-Temporal Parallel Algorithm

The spatial-temporal parallel algorithm is based on the temporal parallel algorithm. The PEs are divided into groups and the temporal parallelism is achieved by assigning different video sections to different PE groups. The compression of each video section is conducted by a PE group in a spatial parallel mode, i.e. each frame in the video section is spatially divided into slices that are compressed in parallel by the PEs in the group. More specifically, all of the PEs are evenly divided into groups, each of which contains n PEs. Among the n PEs in each group, m PEs ($m \leq n$) are used for computation (denoted as *computation PEs*). The rest of the PEs are reserved for special purposes. One of the PEs in a group reads a video section and sends the data to those m computation PEs in the same group through message passing. Each frame in the video section is spatially divided into m slices which are processed by the m computation PEs in parallel. Since each PE on the Paragon has sufficient memory, even though a computation PE processes only a part of each frame, the entire video section is sent to all of the computation PEs. Therefore, we do not have to divide a video section into frames or a frame into several parts and send different parts to different PEs using multiple *send* and *receive* operations. The communication overhead is reduced. According to the number of PEs in a group and the size of the images, each PE is able to determine the portions of data in the entire video section that need to be compressed locally. The compressed data from each computation PE is then

sent to one of the PEs in that group, where the compressed data is assembled and written to the disk (denoted as the output PE).

The data flow in each group of PEs along with the I/O and the communication operations of the algorithm are summarized as follows. A single video section is read by one of the PEs in a group using one *read* operation. Then the video section is sent to all the m computation PEs in the group using one *send* operation. The video section is compressed frame by frame, with each computation PE processing only a part of each frame. The compressed data of each frame is then sent to the output PE. Suppose there are i frames in the video section, then $i \times m$ *send* operations are used by the computation PEs and $i \times m$ *receive* operations are used by the output PE. The output PE assembles the compressed data and writes the compressed data of the entire video section using one *write* operation. Thus, a minimum number of I/O and communications operations are used.

We used two different schemes in our experiments to realize the spatial parallelism.

Scheme 1

A block diagram of the data flow in this scheme is shown in Figure 4.1. In each group of n PEs, one PE is devoted to I/O operations, which is denoted as the I/O node. The rest of the PEs are used for computation. Thus, $m = n - 1$. The I/O node reads a section of video data from the disk, distributes the data to the computation PEs in the group. Each computation PE compresses a part of each of the frames in the video section and send the compressed data back to the I/O node. The I/O node assembles the compressed data and writes it to the disk. A timing diagram of the task modules for each PE in a group is shown in Figure 4.2. In the diagram we assume that $n = 4$ and there are 3 frames in each video section. To minimize the idle time of the computation PEs, the I/O node reads the next video section from the disk before the I/O node starts collecting

compressed data for the current section, i.e. while the computation PEs are compressing the current section. For the same reason, it distributes the video data for the next section to the computation PEs before it writes the compressed data for the current section to the disk.

Scheme 2

A block diagram of the data flow in this scheme is shown in Figure 4.3. In this scheme, no PEs are devoted to I/O operations. Thus, $m = n$. All the PEs conduct the compression task. Meanwhile one of the PEs is in charge of the data input operations and another one is in charge of the data output operations. The PE that is in charge of the data input operations reads the video data from the disk and distributes it to other PEs in the group. All the PEs in the group participate in computation, with each PE processing a part of each frame in parallel. The compressed data is sent to the PE that is in charge of the data output operations, where the data is assembled and written to the disk. A timing diagram of the task modules for each PE in a group is shown in Figure 4.4.

From the timing diagrams, we can see that both Scheme 1 and Scheme 2 have advantages and disadvantages. For Scheme 1, the computation PEs do not have to wait for the video data to be read in. Therefore it is more efficient for the computation PEs. However, since one PE in each group is dedicated to I/O operations, the total number of PEs involved in the compression is reduced. For Scheme 2, each PE is involved in computation. However, when two of the PEs are conducting input and output operations the rest of the PEs in the group have to wait. Hence, for these two schemes there is a trade off between the number of PEs involved in the computation and the amount of idle time for the computation PEs.

4.2 Results

The parallel computers we used for the experiments are the Intel Paragon at Caltech and the Intel Paragon at Purdue. The video sequences used in our experiments are several ITU-R 601 (704×480 pixels per luminance frame) test sequences. From experimentation, we found that the performance of the MPEG compression algorithm is almost invariant when different data sets are used. Thus, here we only present the results obtained from one MPEG test sequence, known as the *football* sequence. The original sequence has 150 frames, subsampled to the 4:2:0 format in the YUV color space [72, 8]. We extended the sequence to more than 10,000 frames by repeating the original sequence. The video sequence was grouped into sections (GOPs), each of which contained 12 frames to be compressed and a reference frame. A frame pattern of IBBPBBIBBPBB was used. In each of our experiments, 4–6 sections were compressed by each group of PEs. The average speed of a group is defined as the number of frames compressed by the group divided by the overall execution time of the group, including the I/O time, the computation time and the communications time. The overall speed was obtained by summing the average speeds of all the groups. The motion vector search algorithm used is the logarithmic algorithm and produces integer pixel motion vectors.

When the spatial-temporal parallel algorithm was used, the number of computation PEs m was set to 2, 4, 8, 15 and 30. The overall speed of compression of these two schemes are shown in Figure 4.5 and Figure 4.11, respectively. The total time is broken down according to 5 different modules—computation, reading the uncompressed data, distributing the uncompressed data among the processors in a group, collecting the compressed data, and writing the compressed data to the PFS. The percentages of the running time consumed by these modules in both schemes are shown in Figures 4.6–4.10 and Figures 4.12–4.16, respectively. Since it is difficult to accurately determine the idle time without introducing significant overhead, the idling mode is not timed individually. Instead, it is included in the time consumed

by other modules. For instance, the time used for data input includes the time the I/O node spends in the I/O queue, and the time used for collecting the compressed data includes the time the I/O node spends on waiting for the incoming compressed data.

Several sets of the parameters of the spatial-temporal parallel algorithm with faster-than-real-time performance is shown in Table 4.1. A maximum compression speed of 43 frames per second was achieved when the Scheme 2 was used and $m = 2$.

Our I/O management methods are so effective that the time used for data input is less than 2% in both schemes (Figure 4.7 and Figure 4.13). One major factor that makes the I/O operations in the spatial-temporal parallel algorithms so much more efficient than that in the temporal only parallel algorithm is that the spatial parallelism greatly reduce the number of processors conducting data input operations simultaneously. The maximum number of processors that perform data input operations is 256, when $m = 2$ and 512 nodes are used in Scheme 2. The distribution of uncompressed data among processors within a group, instead of I/O operations, consumes a large proportion of the running time, especially when m is large. When $m = 30$, the time used for distributing the uncompressed data can be as much as 50% to 70% of the total running time. Thus, keeping m small is a very important factor to increase the efficiency of the algorithm. The percentage of time used for data output in both schemes is very small (Figure 4.8 and Figure 4.14), which is the reason that the I/O queue is not applied to data output.

One may notice that the percentage of the time used for collecting the compressed data is very high when m is small in Scheme 1 (Figure 4.10), which contradicts our intuition. This is caused by the fact that when m is small, a small number of PEs compress a video section in parallel, resulting in a longer computation time to compress a single section. Thus the I/O nodes have to wait in idle longer for the incoming compressed data. Since this waiting time is included in the time used for collecting data, we get a high percentage data collecting time when m is small. This is also the reason why $m = 2$ is not the most efficient mode in Scheme 1.

4.3 Discussion

The advantage of the spatial-temporal parallel algorithm is that a smaller number of processors are allowed to perform I/O operations. Hence it effectively reduces the I/O contention. However one may ask why the spatial parallelism is necessary. Another possible arrangement, the *temporal-temporal* parallelism, can be implemented if we divide the video section read by a group of PEs into smaller sections, each of which contains one or more consecutive frames, and distribute them to different PEs within the group. It should have the same effect on reducing the I/O contention. The problem is that in the temporal-temporal parallelism it is very difficult to maintain the load balance, since the time required to compress I, P and B frames are different. In spatial-temporal parallelism, all the PEs in a group compress different spatial locations of the same frame. Thus the load balance within a group of PEs in the spatial-temporal parallel algorithm is much better maintained than that in the temporal-temporal algorithm. One should notice that the load balance among different groups is not as critical as that within a group, because if one group of PEs uses less time to compress a section, a new section will be assigned to it immediately—no efficiency is sacrificed here.

While real-time compression of ITU-R 601 video sequences can be achieved, there exists a constant delay—the time used to read, compress and write the first video section. The delay depends on the size of the video section and the number of PEs in a group. This makes it undesirable for some applications, such as live digital TV broadcasting. However, it is still very useful for applications such as the generation of a digital video library, in which the throughput, instead of the delay, is the most important issue.

Suppose each group of PEs has 2 processors and each video section has 12 frames, the total number of frames that are needed to load the entire machine (512 PEs) is about 3000 frames, or 100 seconds of video. Usually a video sequence is much longer than 100 seconds. Hence it is not a problem to keep the Paragon in full

operation, which is a problem for spatial only parallelism. In our experiments, all the uncompressed data of a video sequence resides on the PFS before the compression starts. In practical situations, it is not necessary to have all the data ready before the compression. After the first round of loading, as long as the uncompressed data can be provided to the system at a speed no less than the compression speed, the compression process can be sustained without sacrificing the performance.

In our implementation, special arrangements are made for the input and output algorithms. Instead of being saved in a single file, the compressed data is stored in several files. An index list is generated to indicate the order in which the files should be concatenated. This arrangement can be justified as follows. If the storage media is the parallel file system, the file headers and the index list can be viewed as an overhead on the compressed data, which is less than 0.01% of the total compressed data. Thus compression ratio is sacrificed a little to achieve a huge gain in the compression speed. If the compressed data is to be stored outside the parallel file system directly and these files need to be concatenated while they are being compressed, a dedicated system with a buffer and a multiplexer can easily handle this problem. Also in our implementation, the input data stream is divided into sections of video sequence instead of frames, which can be done when the video sequence is loaded to the disk. In practical situations, the input stream may be fed to the parallel computer directly from a digitizer or a network instead of the parallel file system. In this case a multiplexer at the interface between the parallel computer and the network can be used to group the video stream into appropriate sections.

In the experimentation, our algorithm was implemented on the Intel Paragon. The implementation utilized the Paragon message passing library. Only the basic parallel file system operations (*read* and *write*) and message passing operations (*send* and *receive*) were used. Thus our implementation can be easily exported to other MIMD computers with distributed memory and parallel file systems.

Since the MPEG2 main profile main level is very similar to MPEG1, our approach can be easily extended to the encoding of MEPG2.

Table 4.1 Real-time performance using the spatial-temporal parallel algorithm.

scheme	m	number of nodes	frames/second
1	2	510	31.43
1	4	510	33.59
2	2	510	43.64
2	2	382	33.60
2	4	508	40.04
2	4	380	30.70
2	8	504	32.38

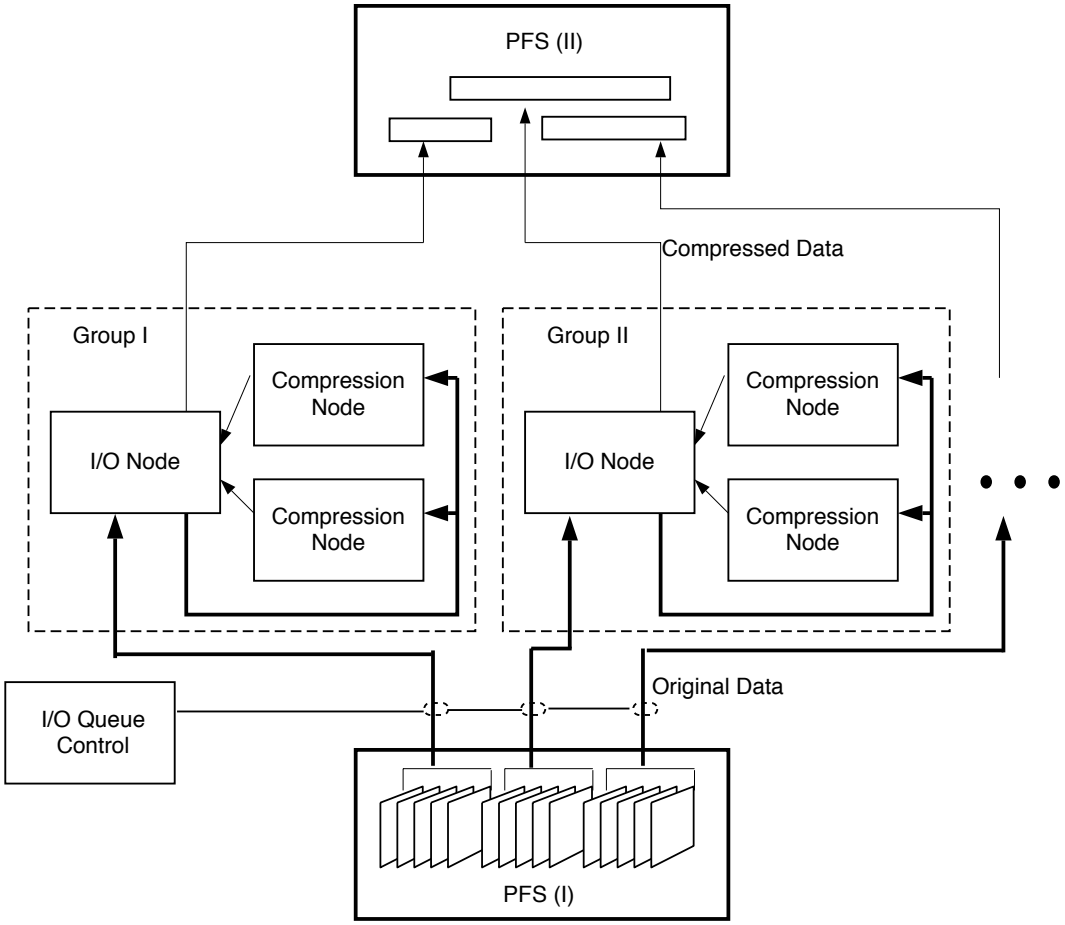


Fig. 4.1. Block diagram of Scheme 1 of the spatial-temporal algorithm.

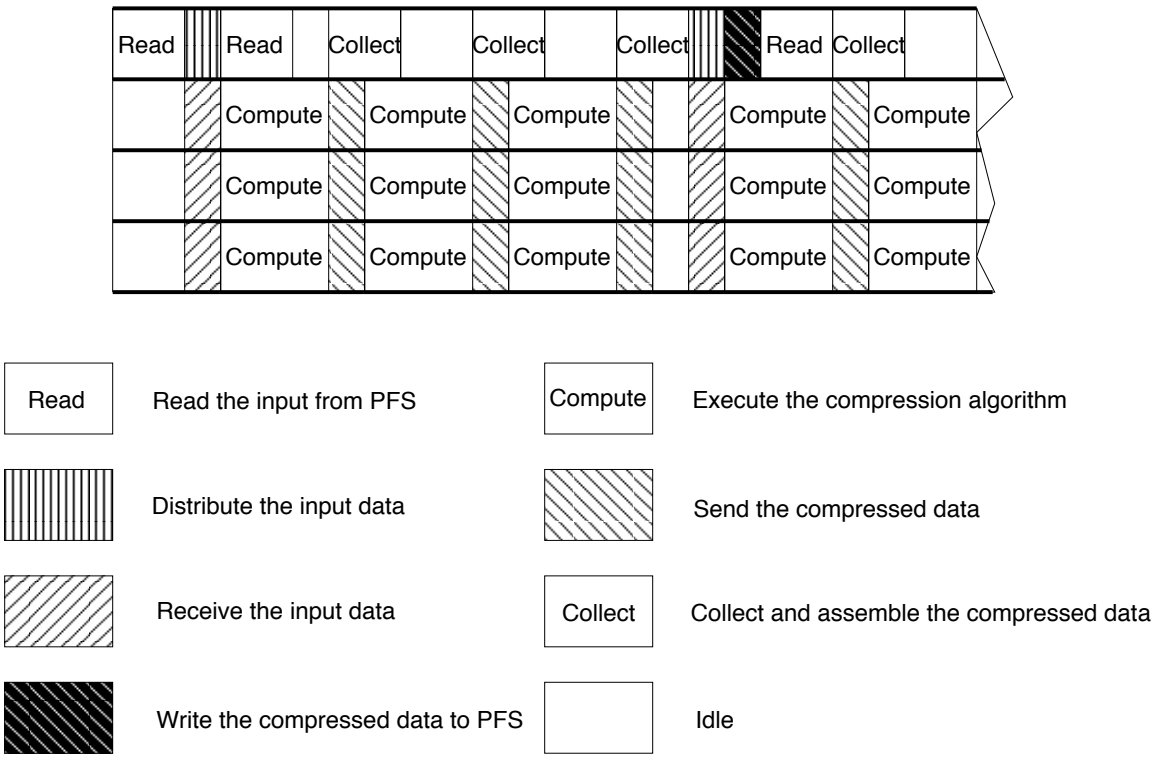


Fig. 4.2. The timing diagram of task modules in Scheme 1. The task modules of a group of 4 PEs are shown. A video section of 3 frames is assumed. Horizontal axis: time, vertical axis: processors.

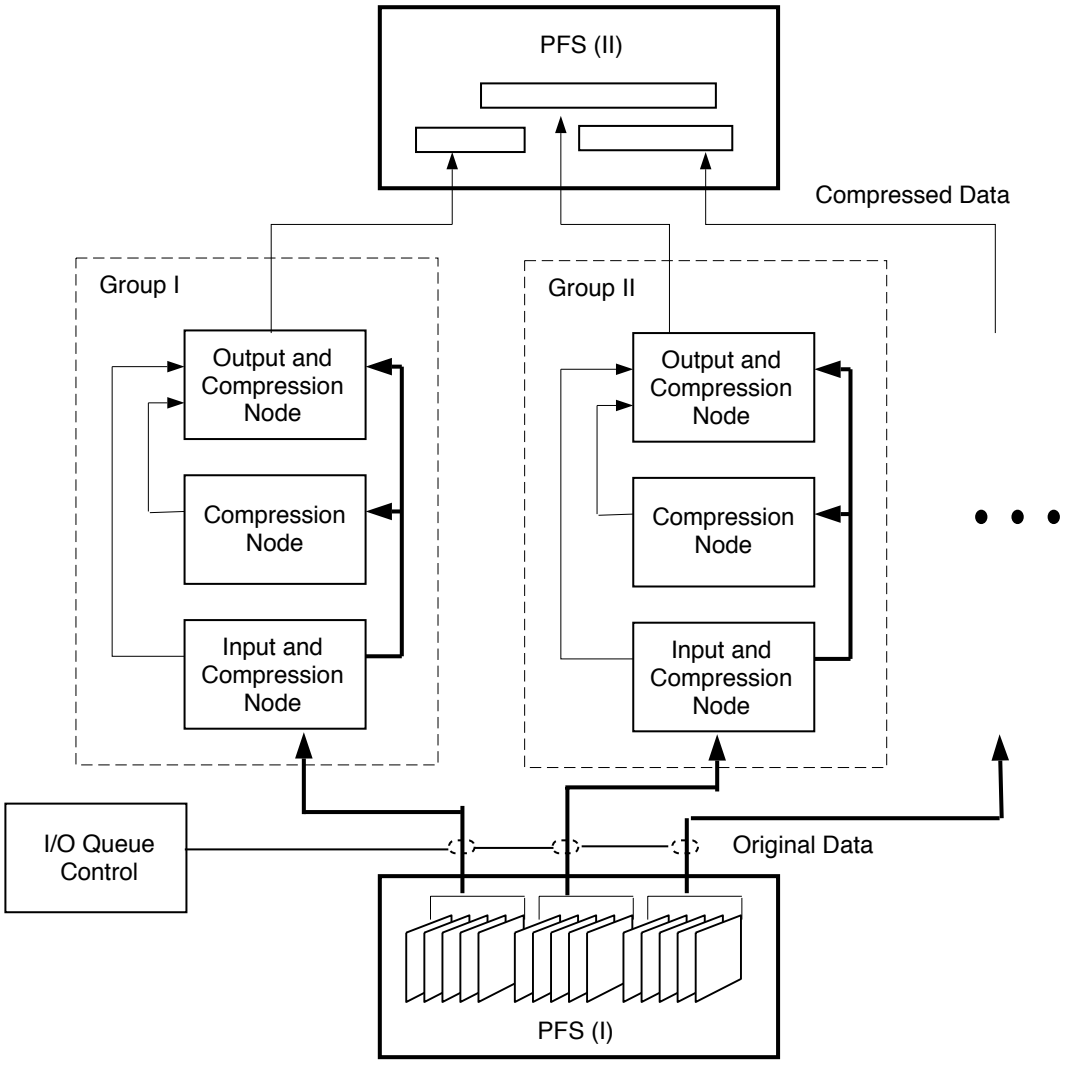


Fig. 4.3. Block diagram of Scheme 2 of the spatial-temporal algorithm.

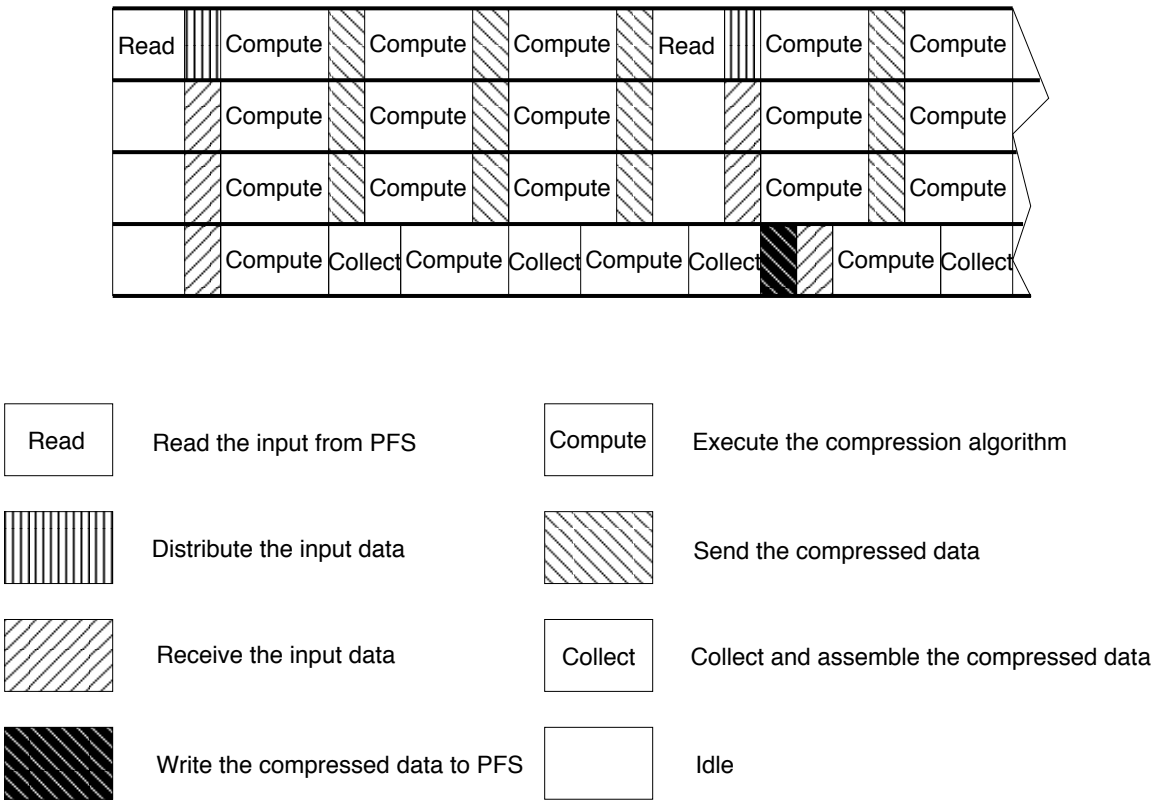


Fig. 4.4. The timing diagram of task modules in Scheme 2. The task modules of a group of 4 PEs are shown. A video section of 3 frames is assumed. Horizontal axis: time, vertical axis: processors.

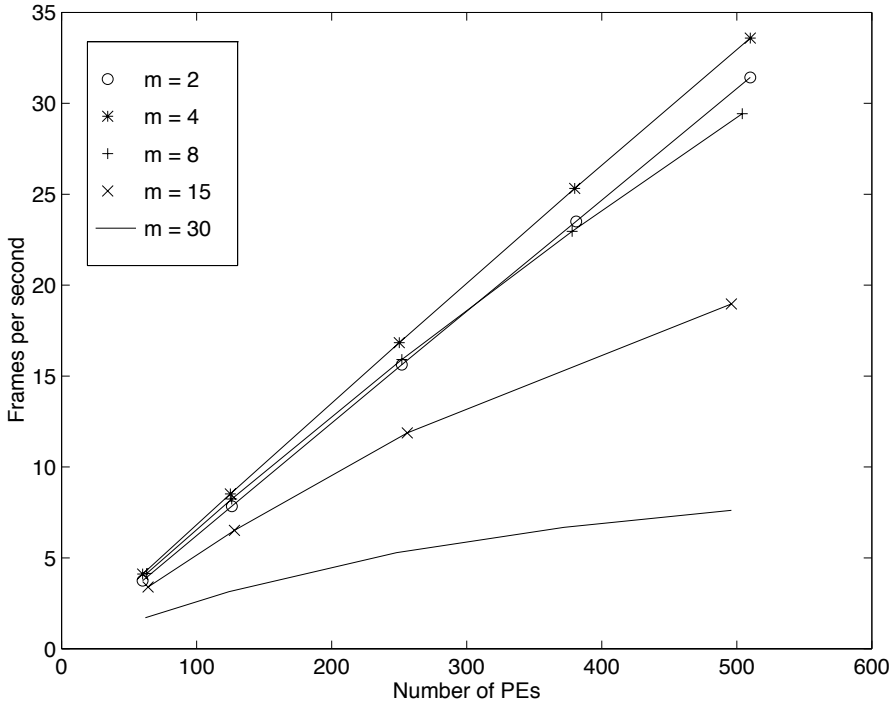


Fig. 4.5. The overall speed for the compression of ITU-R 601 video using spatial-temporal parallelism, Scheme 1.

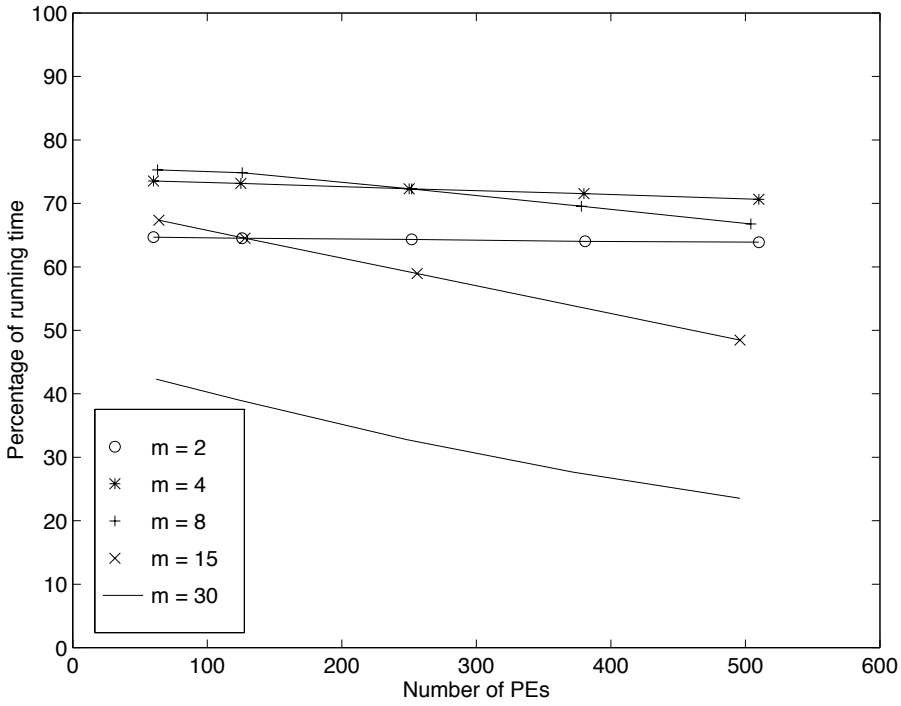


Fig. 4.6. The percentage of time used for computation in the spatial-temporal parallel algorithm, Scheme 1.

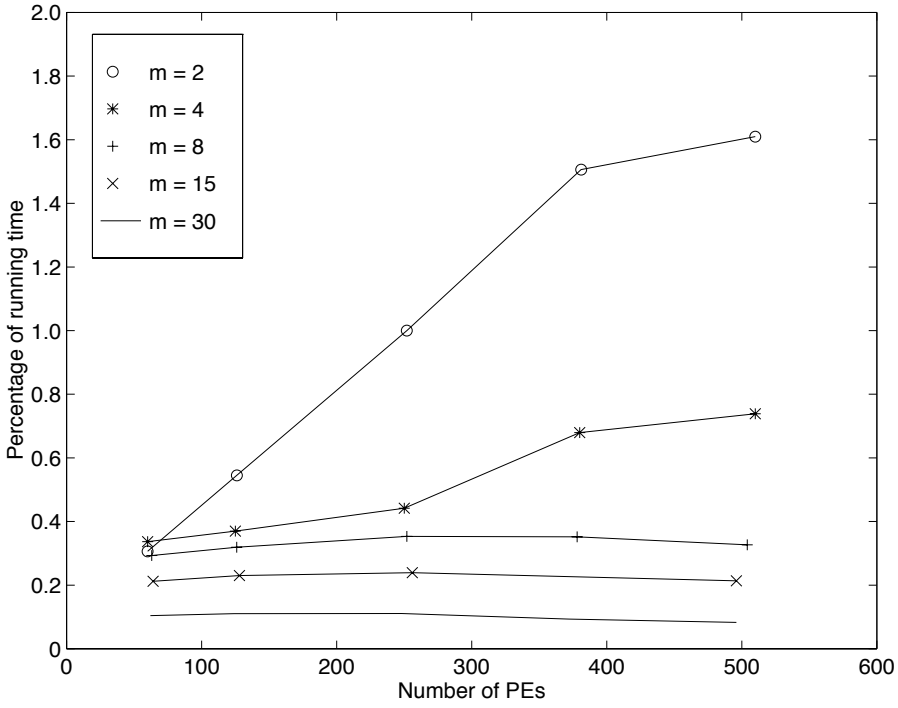


Fig. 4.7. The percentage of time used for data input in the spatial-temporal parallel algorithm, Scheme 1.

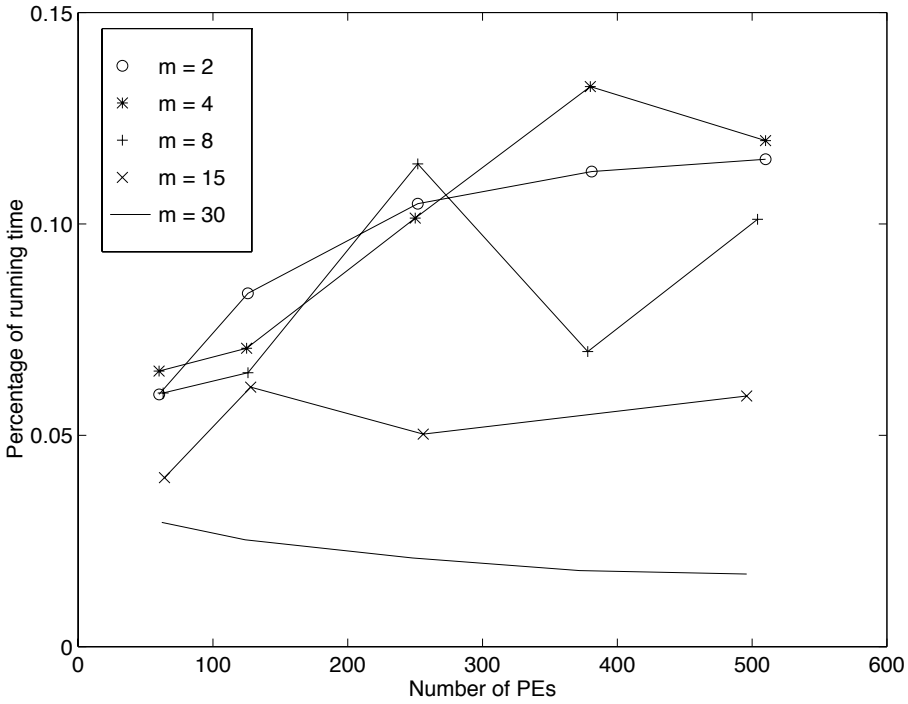


Fig. 4.8. The percentage of time used for data output in the spatial-temporal parallel algorithm, Scheme 1.

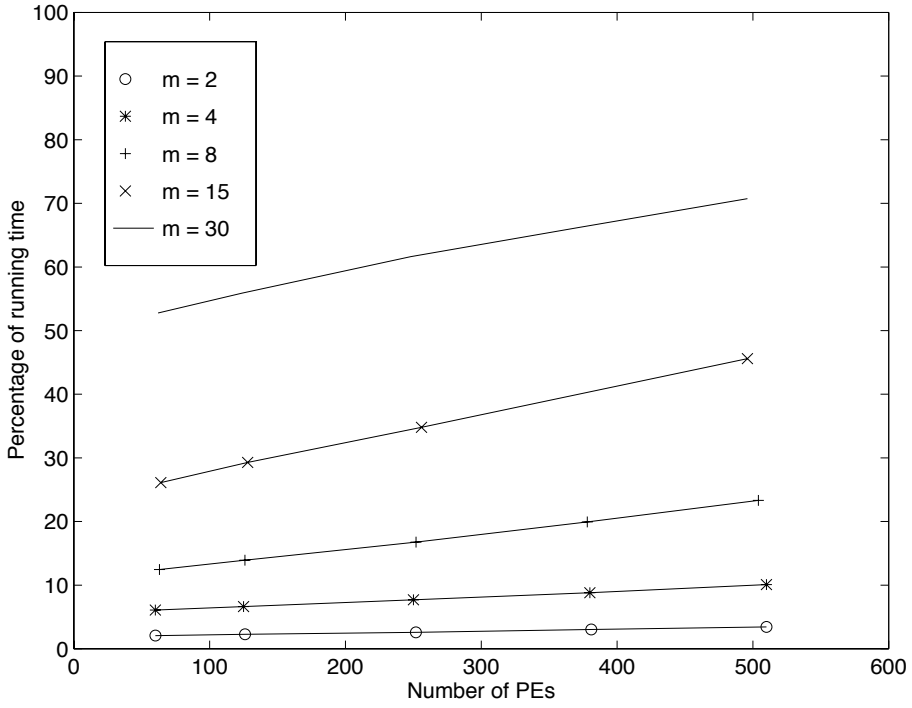


Fig. 4.9. The percentage of time used for distributing the uncompressed data within each group in the spatial-temporal parallel algorithm, Scheme 1.

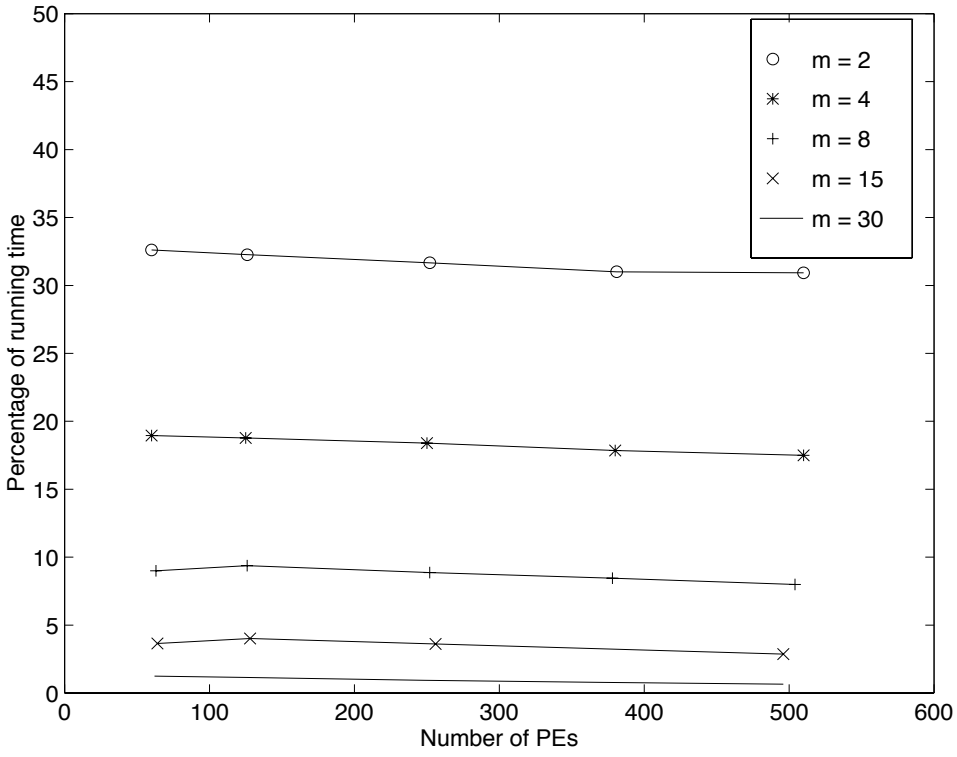


Fig. 4.10. The percentage of time used for collecting the compressed data within each group in the spatial-temporal parallel algorithm, Scheme 1.

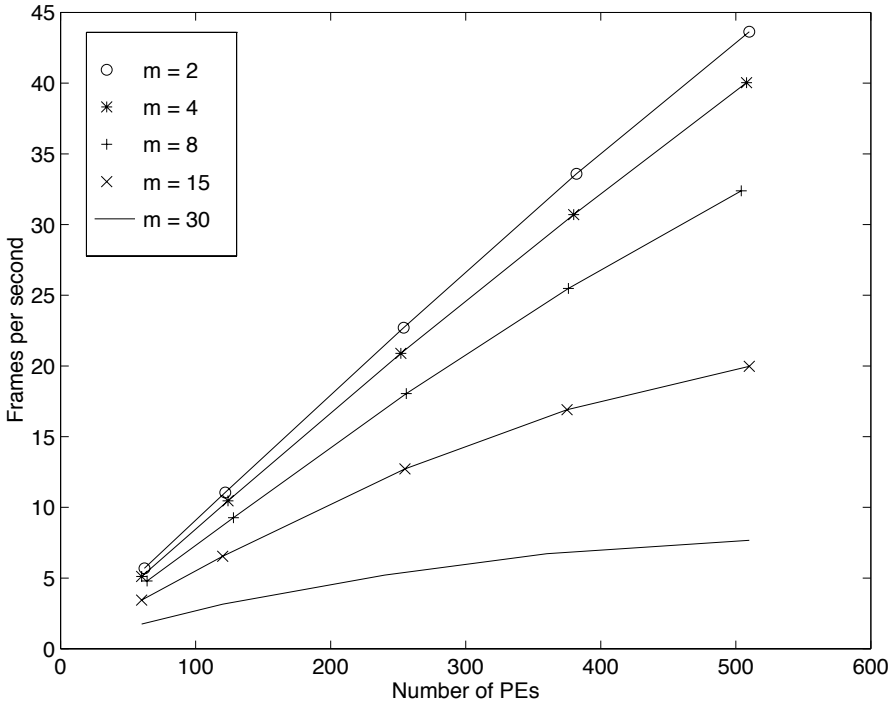


Fig. 4.11. The overall speed of the compression of ITU-R 601 video using spatial-temporal parallelism, Scheme 2.

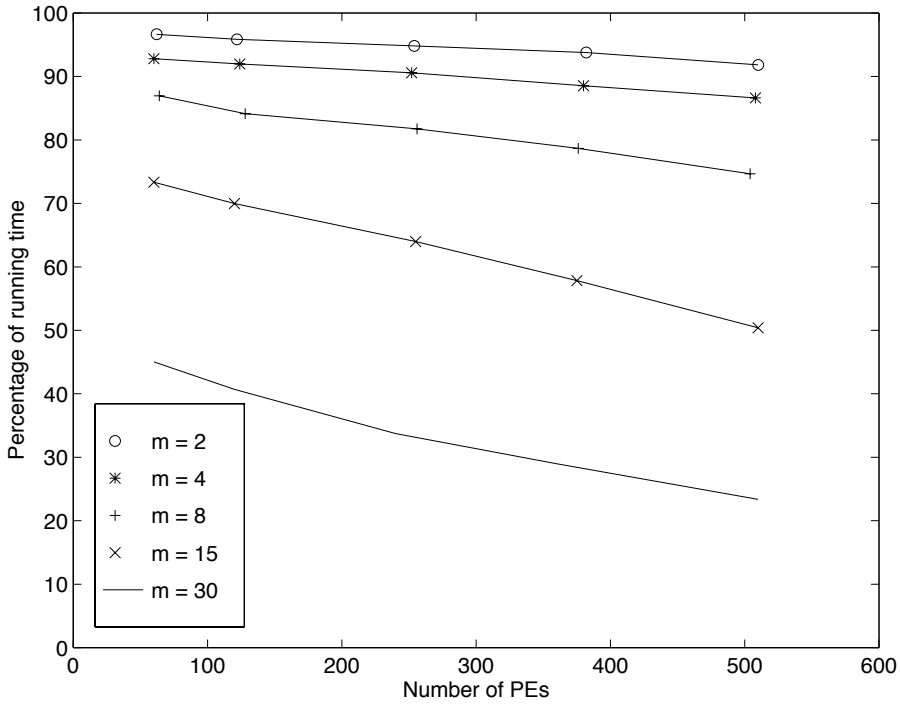


Fig. 4.12. The percentage of time used for computation in the spatial-temporal parallel algorithm, Scheme 2.

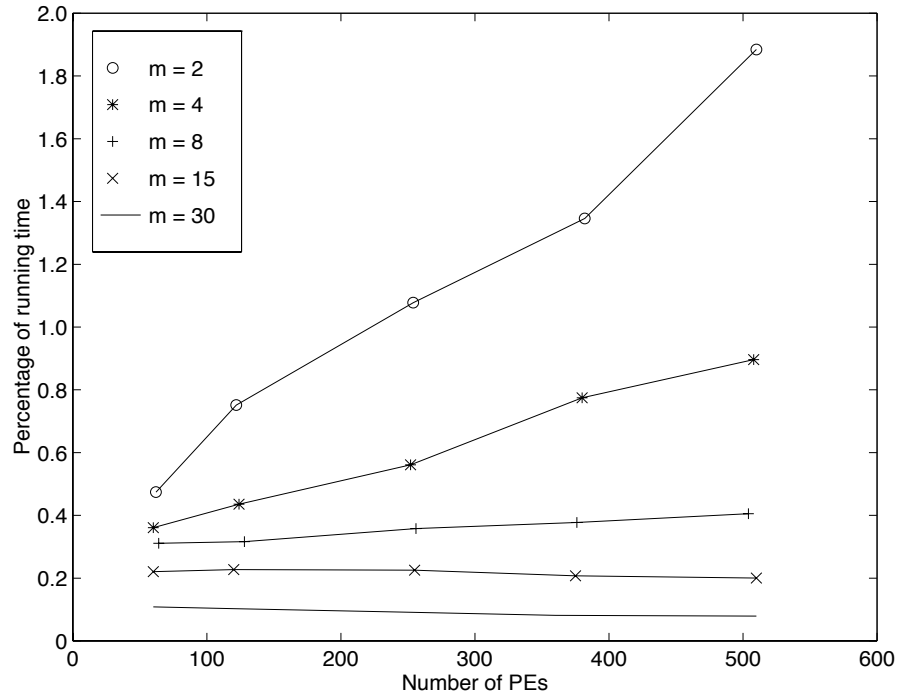


Fig. 4.13. The percentage of time used for data input in the spatial-temporal parallel algorithm, Scheme 2.

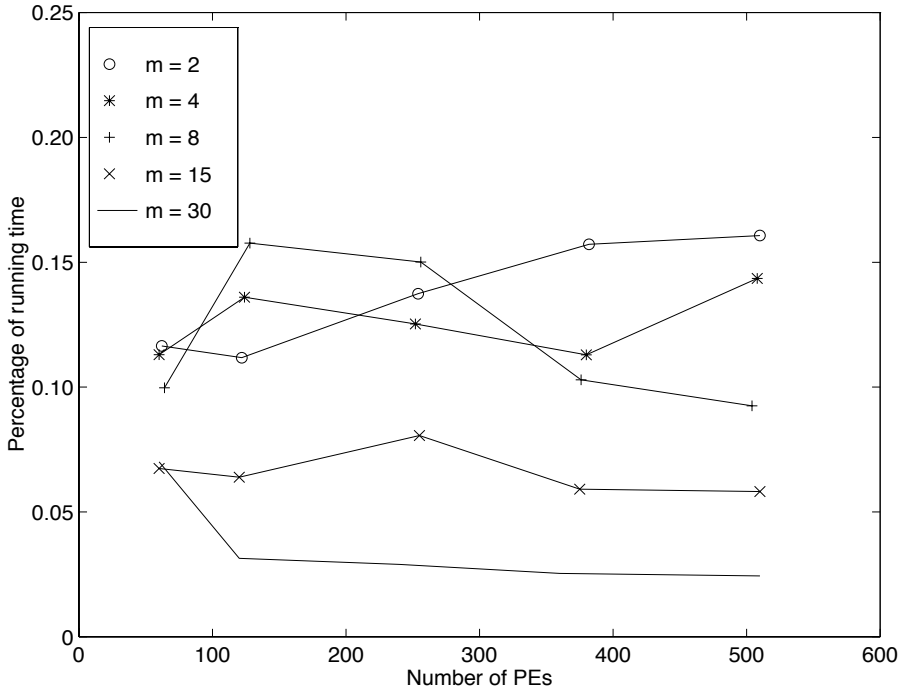


Fig. 4.14. The percentage of time used for data output in the spatial-temporal parallel algorithm, Scheme 2.

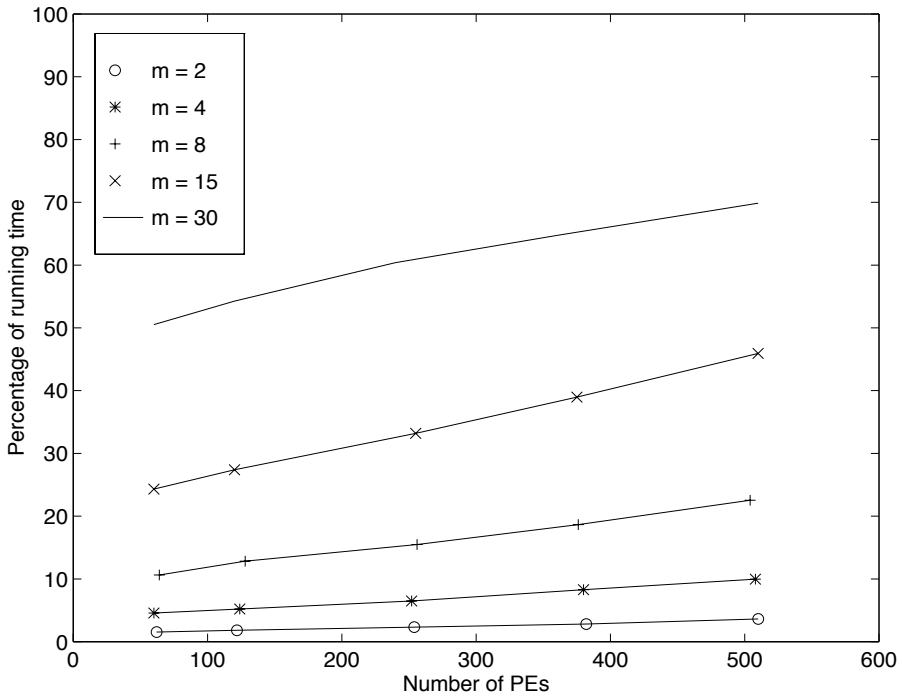


Fig. 4.15. The percentage of time used for distributing the uncompressed data within each group in the spatial-temporal parallel algorithm, Scheme 2.

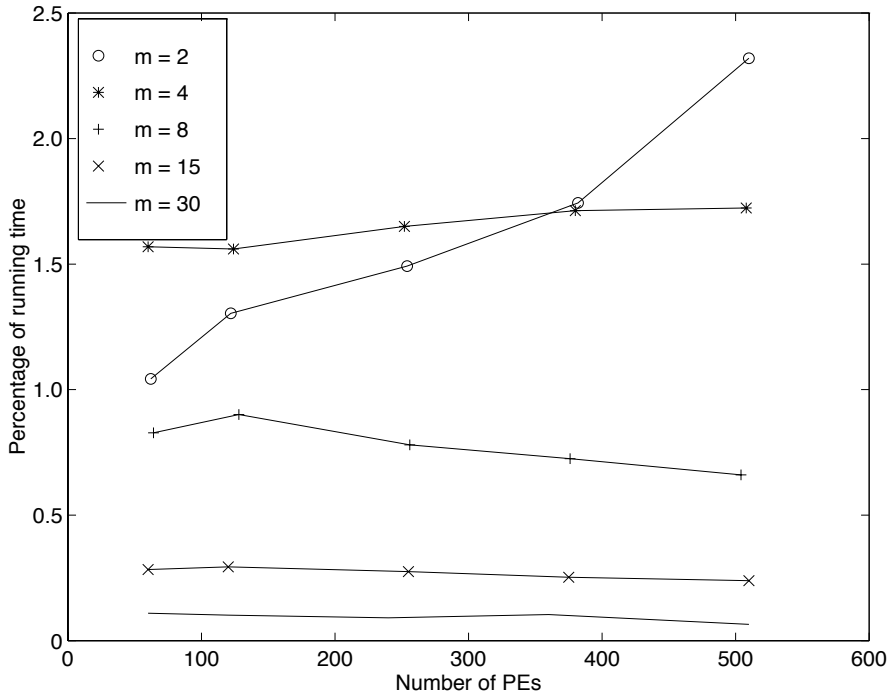


Fig. 4.16. The percentage of time used for collecting the compressed data within each group in the spatial-temporal parallel algorithm, Scheme 2.

5. COLOR EMBEDDED ZEROTREE WAVELET (CEZW): A RATE SCALABLE COLOR IMAGE COMPRESSION TECHNIQUE

Scalability, which includes data rate scalability, spatial resolution scalability, temporal resolution scalability and computational scalability, has become a very important issue in image and video coding. Different applications require different data rates for the compressed image/video and different visual quality (or distortion) for the decompressed image/video. Therefore scalability, the capability of decoding a compressed image or video sequence at different data rates, has become a very important issue in image/video coding. A specific coding strategy known as *embedded rate scalable coding* is well suited for this kind of multi-rate transmission [22]. In embedded coding, all the compressed data is embedded in a single bit stream and can be decoded at different data rates. The decompression algorithm receives the compressed data, from the beginning of the bit stream up to a point when a certain data rate requirement is met. A decompressed image at that data rate can then be reconstructed and the visual quality corresponding to this data rate can be achieved. Thus, to achieve best performance the bits that convey the most important information need to be embedded at the beginning of a compressed bit stream.

Rate scalable image compression, or progressive transmission of images, has been extensively investigated [74, 75, 76]. Reviews on this subject can be found in [77, 78]. Different transforms, such as the Laplacian pyramid [74], the discrete cosine transform (DCT) [76], and the wavelet transform [22, 79], have been used for progressive transmission.

Shapiro introduced a wavelet based embedded rate scalable image compression algorithm. A spatial-orientation tree (SOT), known as a zerotree, was used to exploit

the interdependence between the subbands of a wavelet decomposed image [22]. Since then, variations of the algorithm have been proposed [80, 79, 81, 82], among which the one proposed by Said and Pearlman [79] has attracted the most attention. In [22] and [79], slightly different tree structures were used and good rate scalability was achieved. However, both algorithms were developed for grayscale images. For color images, the algorithm has to be used separately for the three color components. Thus, the interdependence between the color components cannot be exploited. Taubman and Zakhor's subband coding algorithm used a *Layered DPCM* scheme to exploit the correlation between the color components as well as the correlation between different frequency bands [83]. However, the statistical correlations between the color components and between the frequency bands are not significant when subband coding and LC color spaces are used. For most color images, the spatial locations that show large transitions in chrominance components also show large transitions in luminance component, which can be exploited in image compression [84, 85]. In this chapter we present a modified zero-tree wavelet image compression scheme that exploits the interdependence between the color components.

5.1 Embedded Zerotree Wavelet Image Coding

A wavelet transform corresponds to two sets of analysis/synthesis filters, $g/\tilde{g}$ and $h/\tilde{h}$, where g can be treated as a high pass filter and h can be viewed as a low pass filter. By using the filters g and h, the image can be decomposed into four bands. Subsampling is used to translate the subbands to a baseband image. This is the first level of the wavelet transform (Figure 5.1). Usually this transform can be repeated to the low-low (LL) band. Thus, a typical 2-D discrete wavelet transform used in image processing will generate a hierarchical pyramidal structure shown in Figure 5.2. The inverse wavelet transform is achieved by reversing the transform process and replacing the analysis filters with the synthesis filters and down-sampling with up-sampling (Figure 5.3). The wavelet transform can decorrelate the image pixel values and result

in frequency and spatial-orientation separation. The transform coefficients in each band exhibit unique statistical properties that can be used for encoding the image.

For image compression, quantizers can be designed specifically for each band. The quantized coefficients can then be entropy coded using either Huffman coding or arithmetic coding [86, 87, 88]. In embedded coding, a key issue is to embed the more important information at the beginning of the bit stream. From a rate-distortion point of view, one wants to quantize the coefficients that cause larger distortion first. Let the wavelet transform be $\boldsymbol{c} = T(\boldsymbol{p})$, where $\boldsymbol{p}$ is the collection of image pixels and $\boldsymbol{c}$ is the collection of transform coefficients. The reconstructed image $\hat{\boldsymbol{p}}$ is obtained by the inverse transform $\hat{\boldsymbol{p}} = T^{-1}(\hat{\boldsymbol{c}})$, where $\hat{\boldsymbol{c}}$ is the quantized transform coefficients. The distortion introduced in the image is $D(\boldsymbol{p} - \hat{\boldsymbol{p}}) = D(\boldsymbol{c} - \hat{\boldsymbol{c}}) = \sum_i D(c_i - \hat{c}_i)$, where $D(\cdot)$ is the distortion metric and the summation is over the entire image. The greatest distortion reduction can be achieved if the transform coefficient with the largest magnitude is coded with infinite precision. Thus attempts have been made to encode the transform coefficients with larger magnitudes first. Furthermore, to strategically distribute the bits such that the decoded image will look "natural", instead of spending a lot of bits coding one coefficient precisely, progressive refinement or bit-plane coding is used. Hence, in the coding procedure multiple passes through the data are made. Let C be the largest magnitude in $\boldsymbol{c}$. In the first pass, those transform coefficients with magnitudes greater than $\frac{1}{2}C$ are considered significant and are quantized to $\frac{3}{4}C$. The rest are quantized to 0. In the second pass, those coefficients that have been quantized to 0 but have magnitudes in between of $\frac{1}{4}C$ and $\frac{1}{2}C$ are considered significant and are quantized to $\frac{3}{8}C$. Again the rest are quantized to zero. Also those significant coefficients in the last pass are refined to one more level of precision, i.e. $\frac{5}{8}C$ or $\frac{7}{8}C$. This process can be repeated until the data rate meets the requirement or the quantization step is small enough. Thus, we can achieve the largest distortion reduction with the smallest number of bits, while the coded information is distributed across the image.

However, to make this strategy work we need to code the position information along with the magnitude information of the wavelet coefficients. It is critical that the positions of the significant coefficients be coded efficiently. One could scan the image in a given order that is known to both the encoder and decoder. A coefficient is coded 0 if it is insignificant or 1 if it is significant relative to the threshold. However observations have shown that the majority of the transform coefficients are insignificant when compared to the threshold, especially when the threshold is high. These coefficients will be quantized to zero, which will not reduce the distortion. However, we still have to use at least one symbol to code each of them. Using more bits to code these insignificant coefficients results in lower efficiency. It has been observed that coefficients which are quantized to zero at a certain pass have structural similarity across subbands in the same spatial orientation. Thus spatial-orientation trees (SOTs) can be used to quantize large areas of insignificant coefficients efficiently (e.g. zerotree in [22]).

The algorithms proposed by Shapiro (EZW) [22], and Said and Pearlman (SPIHT) [79] use slightly different SOTs (shown in Figure 5.4). The major difference between these two algorithms lies in the fact that they use different strategies to scan the transformed pixels. Comparing the results of SPIHT and EZW, one can see that the SOT used by Said and Pearlman [79] is more efficient than Shapiro's [22].

5.2 Embedded Coding of Color Images

Many wavelet based rate scalable algorithms, such as EZW [22] and SPIHT [79],were developed for grayscale images. To code a color image, the color components are treated as three individual grayscale images and the same coding scheme is used for each component [82]. The interdependence between the color components is not exploited. To exploit the interdependence between color components, the algorithm may also be used on the decorrelated color components generated by a linear transform. In Said and Pearlman's algorithm [79], the Karhunen-Loeve (KL) transform is used [89]. The KL transform is optimal in the sense that the transform coefficients

are uncorrelated. The KL transform, however, is image dependent, i.e. the transform matrix needs to be obtained for each image and transmitted along with the coded image. Also the bits allocation for each color component has to be determined before the compression [82].

The red-green-blue (RGB) color space is commonly used because it is compatible with the mechanism of color display devices. However, the color components in the RGB space are significantly correlated. Other color spaces are used, among these are the luminance and chrominance (LC) spaces which are popular in video/television applications. An LC space, e.g., YCrCb, YUV or YIQ, consists of a luminance component and two chrominance (color difference) components. The LC spaces are popular because the luminance signal can be used to generate a grayscale image, which is compatible with monochrome systems, and the three color components have little correlation, which facilitates the encoding and/or modulation of the signal [90, 91].

Although the three components in a LC space are uncorrelated, they are not independent. Observations have shown that at the spatial locations where chrominance signals have large transitions, the luminance signal also has large transitions [84, 85]. Transitions in an image usually correspond to coefficients with large magnitudes in high frequency bands of the wavelet transformed image. Thus, if a transform coefficient in a high frequency band of the luminance signal has small magnitude, the transform coefficient of the chrominance components at the corresponding spatial location and frequency band should also have small magnitude [92, 93]. In embedded zerotree coding, if a zerotree occurs in the luminance component, a zerotree at the same location in the chrominance components is highly likely to occur. This interdependence of the transform coefficients signals between the color components can be exploited.

In our new algorithm, the YUV space is used. We refer to this approach as *Color Embedded Zerotree Wavelet (CEZW)*. The SOT is established as follows: The original SOT structure in Shapiro's algorithm is used for all three color components. Besides that, each chrominance node is also a child node of the luminance node of the same

location. Thus each chrominance node has two parent nodes: one is of the same chrominance component in a lower frequency band, and the other is of the luminance component. A diagram of the SOT is shown in Figure 5.5.

In *CEZW*, the coding strategy is similar to Shapiro's algorithm [22]. The algorithm also consists of dominant passes and subordinate passes. The symbols used in the dominant pass are positive significant, negative significant, isolated zero and zerotree. In the dominant pass, the luminance component is first scanned. For each luminance pixel, all descendents, including those of the luminance component and those of the chrominance components, are examined and appropriate symbols are assigned. The zerotree symbol is assigned if the current coefficient and its descendents in the luminance and chrominance components are all insignificant. The two chrominance components are alternately scanned after the luminance component is scanned. The coefficients in the chrominance that have already been encoded as part of a zerotree while scanning the luminance component are not examined. The subordinate pass, which is the refinement of the coefficients that have been coded as significant in previous passes, is essentially the same as that in Shapiro's algorithm. A summary of *CEZW* is shown in Figure 5.6.

5.3 Results and Discussion

In our experiments, the original images are in YUV 4:1:1 format. For the SPIHT[1] and JPEG algorithms, the images are converted to RGB 4:4:4 format from the YUV 4:1:1 format. In our experiments, the wavelet decomposition was based on Daubechies' 9-7 tap filter bank [94]. Adaptive arithmetic coding is used as the entropy encoder [87, 88]. The size of the image, the levels of wavelet transform, the initial threshold T, and the maximum data rate is embedded at the beginning of the bit stream as the header information. The image is decoded at different data rates within the range of 0.5 bits per pixels (bpp) to 1.5 bpp. The comparison of decoded

[1]The demonstration program was obtained from Said and Pearlman's web page at `http://ipl.rpi.edu:80/SPIHT`.

images using *CEZW*, JPEG, and SPIHT at 0.5 bpp is shown in Figures 5.7–5.11[2]. Enhanced difference images between the original images and the decoded images using *CEZW*, JPEG, and SPIHT at 0.5 bpp are shown in Figures 5.12–5.16. Our subjective experiments have shown that *CEZW* produces images with better quality than that from Said and Pearlman's SPIHT as well as JPEG at the same data rate. The peak signal-to-noise (PSNR) ratios from *CEZW* are compared with that from SPIHT and JPEG in Table 5.1, where the PSNR is obtained as

$$10 \log_{10} \frac{255^2}{(MSE(Y) + MSE(U) + MSE(V))/3}$$

or

$$10 \log_{10} \frac{255^2}{(MSE(R) + MSE(G) + MSE(B))/3}$$

for YUV or RGB color spaces, respectively. The PSNR from *CEZW* is 4 dB higher than that from SPIHT at all data rates. Here we want to point out that the PSNR numbers are possibly biased in favor of *CEZW* since the YUV color space is used as the base color space.

CEZW does not require image-dependent transforms such as KL transforms used in Said and Pearlman's algorithm to decorrelate color components. A spatial-orientation tree that links not only the frequency bands but also the color channels is used for scanning the wavelet coefficients, such that the interdependence between different color components in LC spaces is automatically exploited. Also there is no need for specifically allocating the bits among color components because of the alternate scanning of all the color components. Since *CEZW* is designed for color spaces with LC components, it can be readily applied to the rate scalable video compression algorithm, which is presented in the next chapter.

[2]Only the grayscale images are reproduced in this thesis. The color images are available at `ftp://skynet.enc.purdue.edu/pub/dist/delp/shen-thesis`.

Table 5.1 PSNR of decoded images using *CEZW*, SPIHT and JPEG.

images		*girls*	*lenna*	*model*	*peppers*	*tiger*
0.5 bpp	*CEZW*	36.1	36.9	37.4	41.9	31.9
	SPIHT	32.3	33.3	33.2	37.8	27.9
	JPEG	29.7	30.3	30.8	34.6	25.1
1.0 bpp	*CEZW*	39.3	40.1	40.5	44.8	35.1
	SPIHT	35.7	35.9	36.1	39.8	31.4
	JPEG	32.9	33.5	33.8	37.2	28.8
1.5 bpp	*CEZW*	42.3	42.7	43.0	45.9	38.3
	SPIHT	38.0	37.6	38.2	41.3	34.0
	JPEG	35.0	35.2	35.7	38.4	31.2

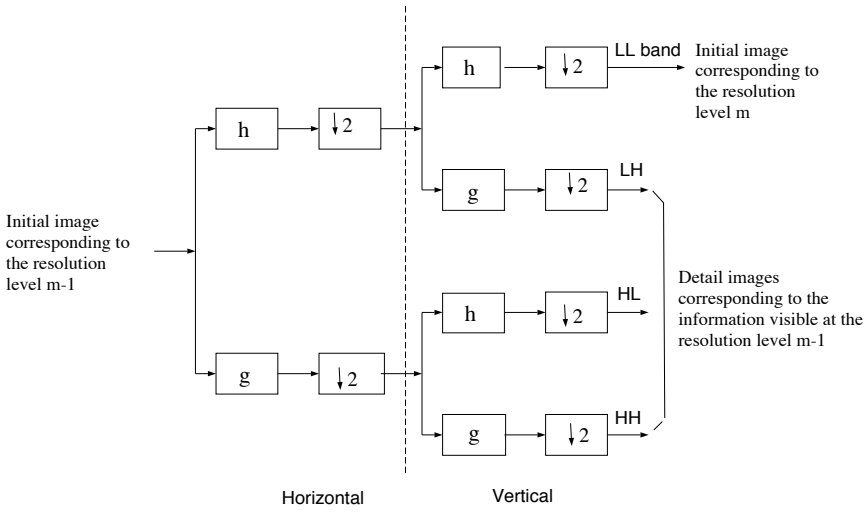


Fig. 5.1. One level of the wavelet transform.

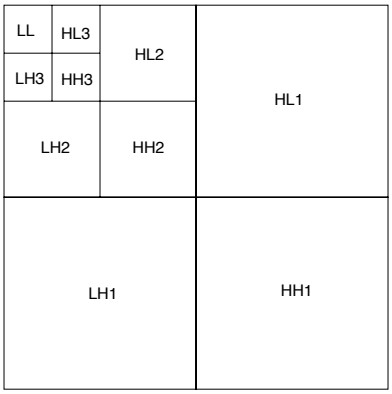


Fig. 5.2. Pyramid structure of a wavelet decomposed image. Three levels of the wavelet decomposition are shown.

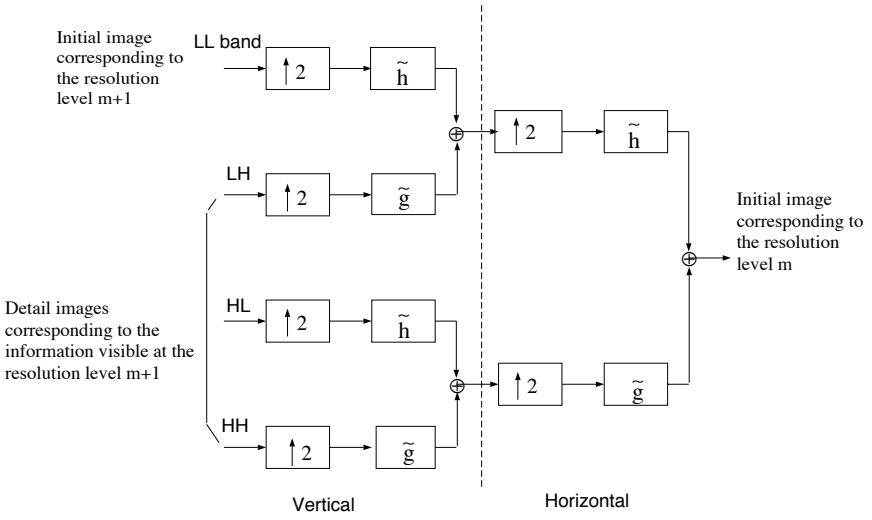
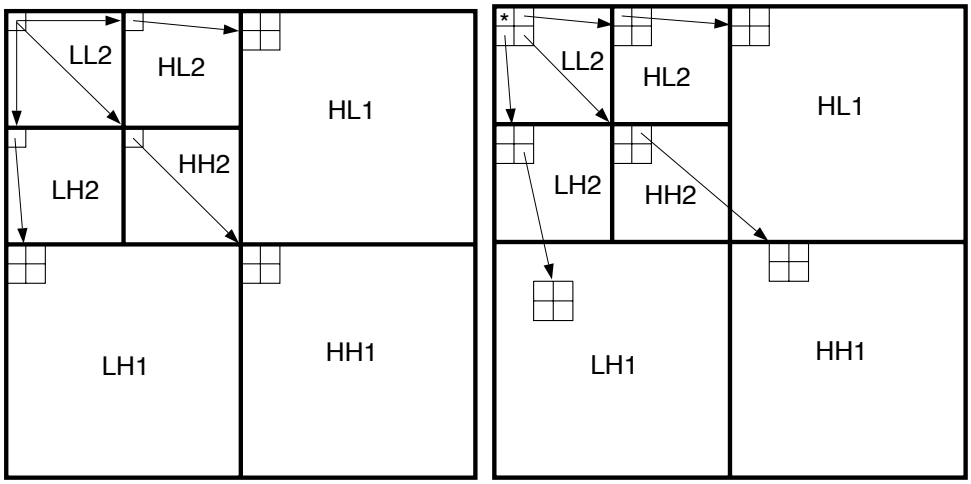


Fig. 5.3. One level of the inverse wavelet transform.



(a) (b)

Fig. 5.4. Diagrams of the parent-descendent relationships in the spatial-orientation trees. (a) Shapiro's algorithm. Notice that the pixel in the LL band has 3 children. Other pixels, except for those in the highest frequency bands, have 4 children. (b) Said and Pearlman's algorithm. One pixel in the LL bands (noted with "*") does not have a child. Other pixels, except for those in the highest frequency bands, have 4 children.

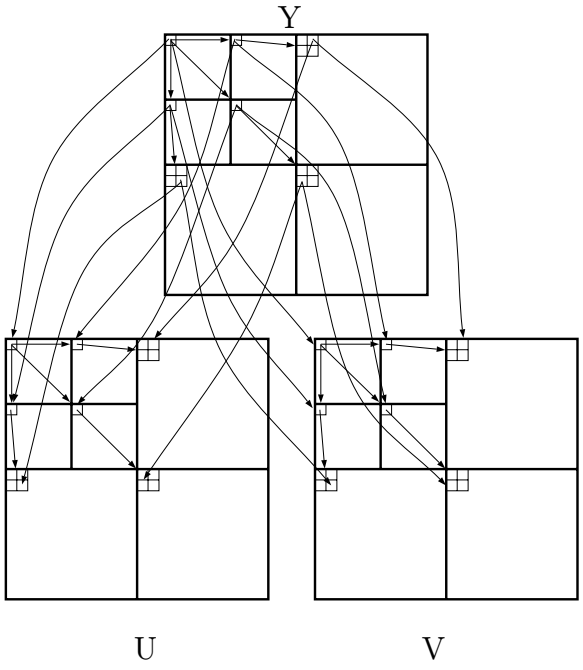


Fig. 5.5. Diagram of the parent-descendent relationships in the *CEZW* algorithm. This tree is developed on the basis of the tree structure in Shapiro's algorithm. The YUV color space is used.

1. Wavelet transform is performed on each of the three color components separately. Let T be the largest magnitude in wavelet transform coefficients $\boldsymbol{c}$.

2. While bit budget is not exhausted

   (a) $T = \frac{1}{2}T$.

   (b) Dominant pass:

      i. The Y component is scanned and for each Y node its children in Y component as well as those in U and V components are compared with T. Symbols of *Positive significant (POS)*, *Negative significant (NEG)*, *Zerotree (ZT)* and *Isolated Zero (IZ)* are assigned and entropy coded.

      ii. The U and V components are alternately scanned. The coefficients and their children nodes are compared with T. Those coefficients that have been coded as part of a zerotree in step (i) are not examined. Symbols of POS, NEG, ZT and IZ are assigned and entropy coded.

   (c) Subordinate pass

      i. The coefficients that have been coded as significant in previous passes (excluding the dominant pass just preceding this subordinate pass) are examined. The quantization error is compared with T and symbols of *Significant (SIG)* and *Insignificant (INS)* are assigned and entropy coded.

Fig. 5.6. A description of *CEZW*.

(Original) (JPEG)

(SPIHT) (*CEZW*)

Fig. 5.7. The original and the decoded images of *Girls* at 0.5 bpp using *CEZW*, JPEG and SPIHT.

(Original) (JPEG)

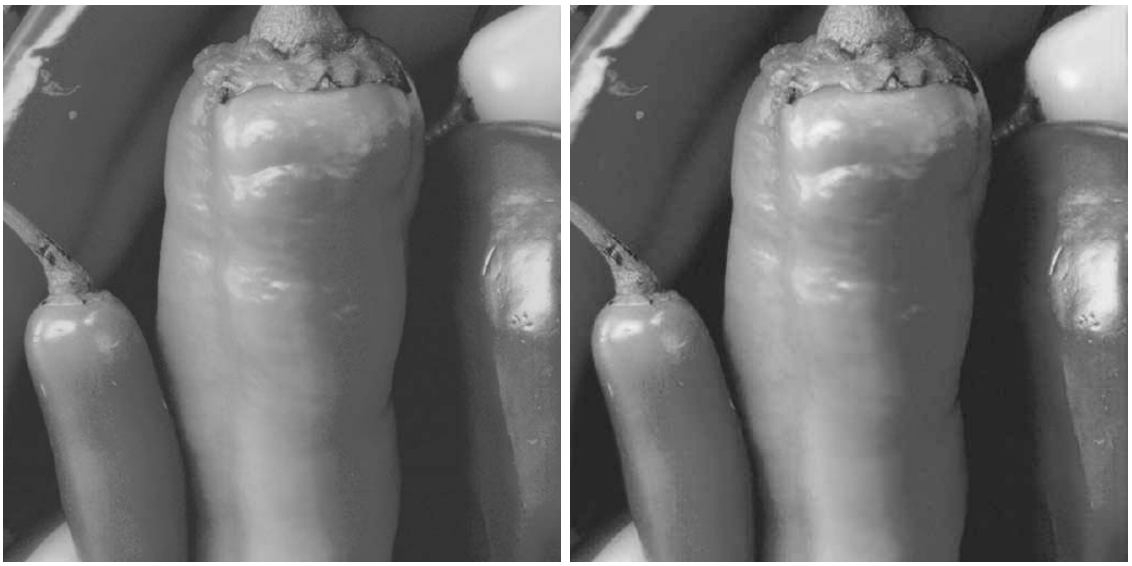
(SPIHT) (*CEZW*)

Fig. 5.8. The original and the decoded images of *Lenna* at 0.5 bpp using *CEZW*, JPEG and SPIHT.

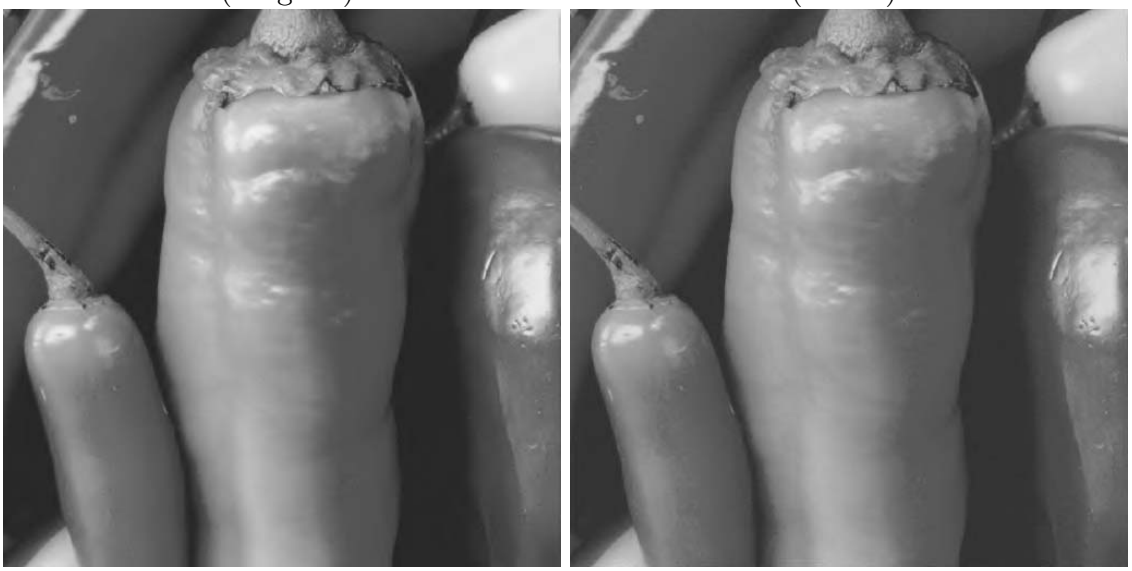
(Original) (JPEG)

(SPIHT) (*CEZW*)

Fig. 5.9. The original and the decoded images of *Model* at 0.5 bpp using *CEZW*, JPEG and SPIHT.

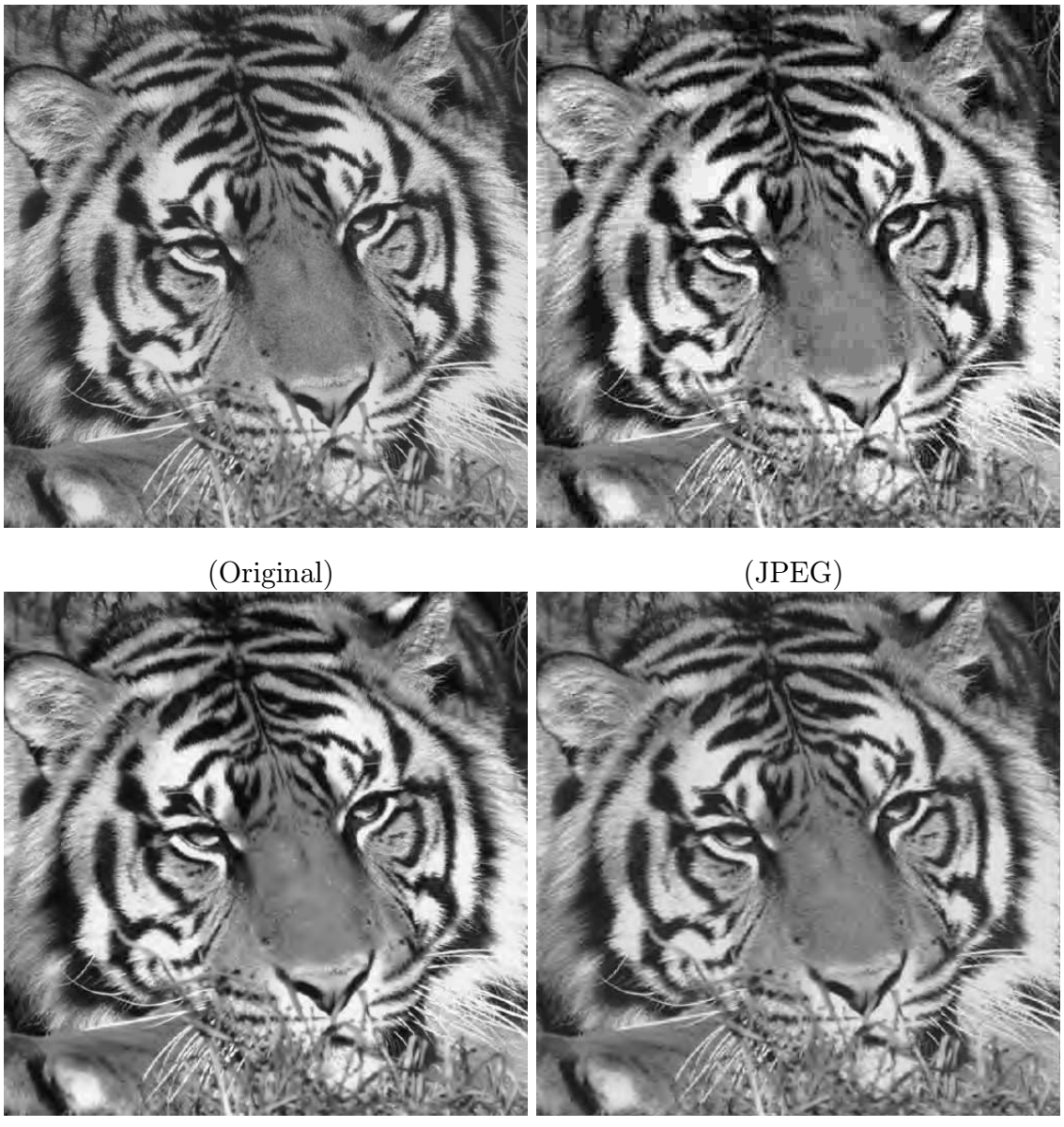


(Original) (JPEG)



(SPIHT) (*CEZW*)

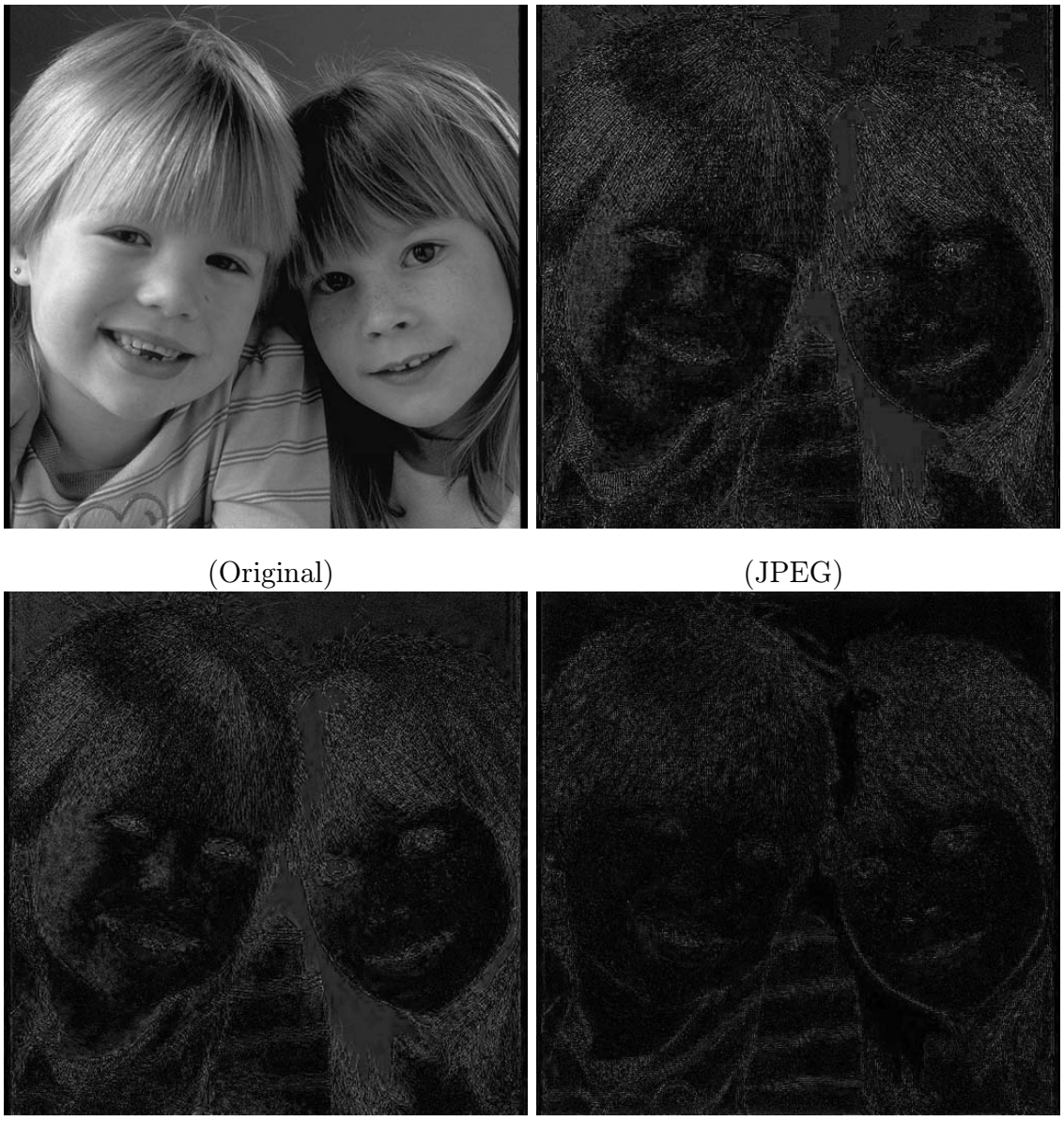
Fig. 5.10. The original and the decoded images of *Peppers* at 0.5 bpp using *CEZW*, JPEG and SPIHT.



(Original) (JPEG)

(SPIHT) (*CEZW*)

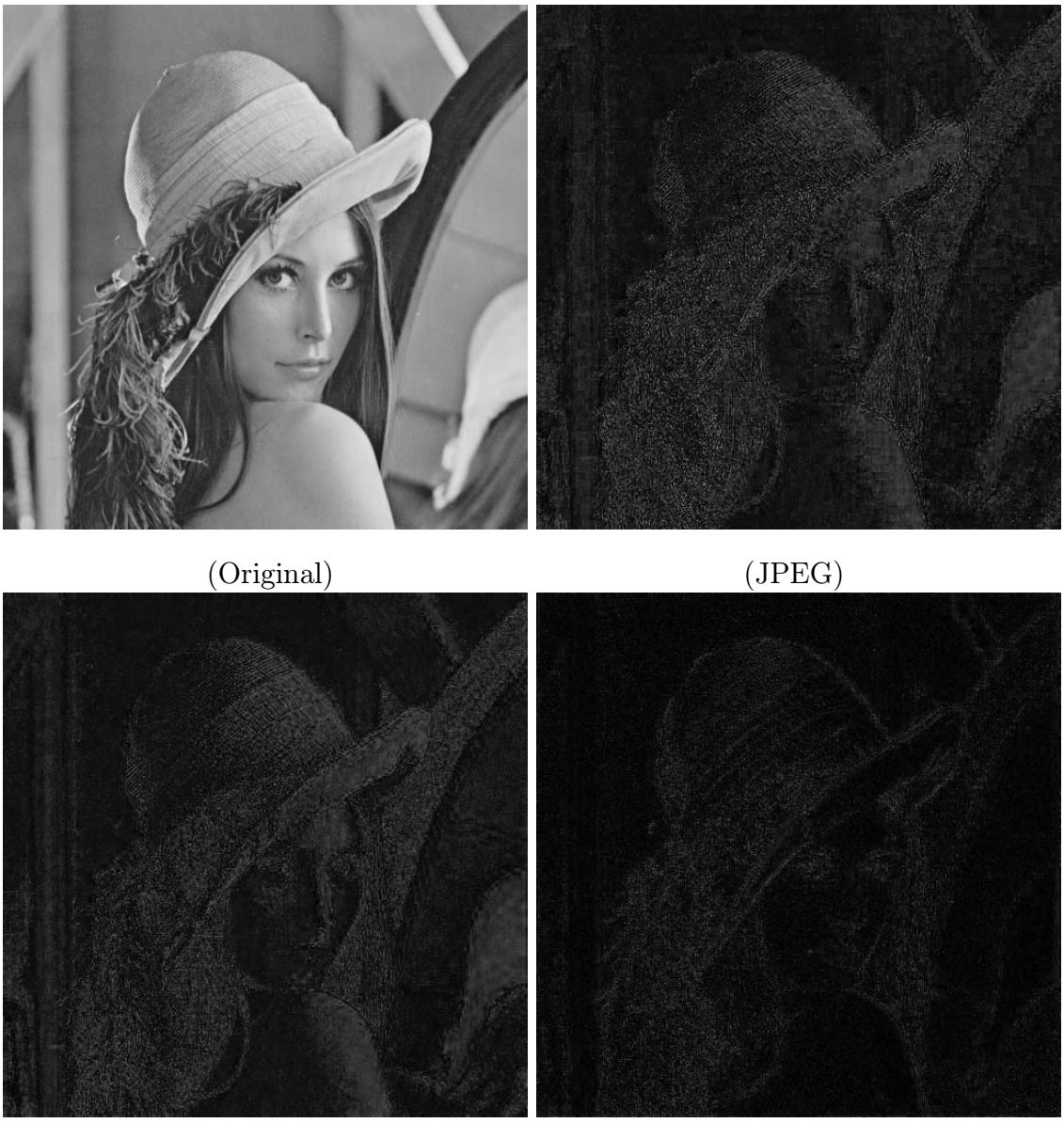
Fig. 5.11. The original and the decoded images of *Tiger* at 0.5 bpp using *CEZW*, JPEG and SPIHT.



(Original) (JPEG)

(SPIHT) (*CEZW*)

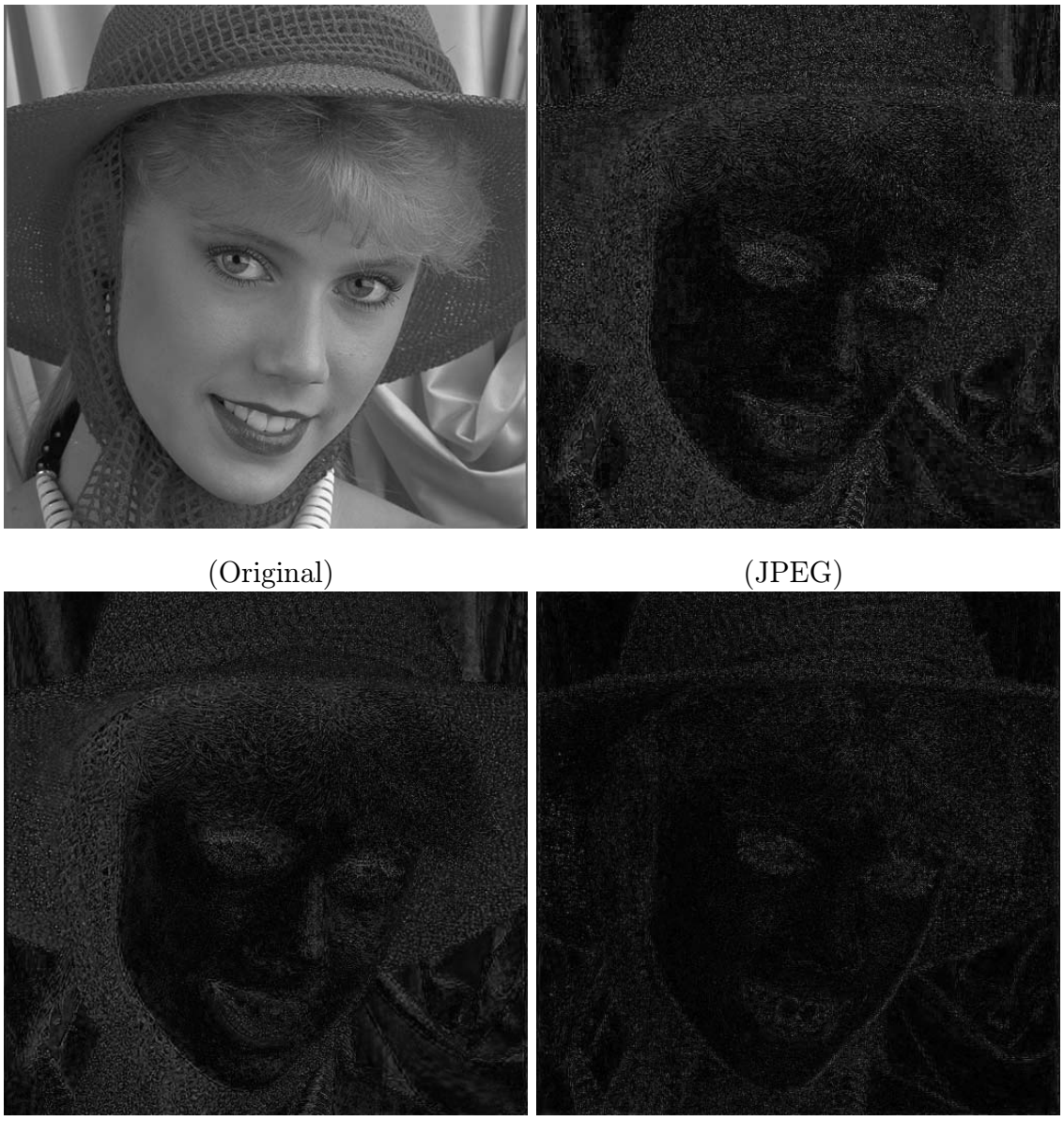
Fig. 5.12. The original and the difference images of *Girls* at 0.5 bpp using *CEZW*, JPEG and SPIHT. The difference images are enhanced by a factor of 3.64 to show the coding artifacts.



(Original) (JPEG)

(SPIHT) (*CEZW*)

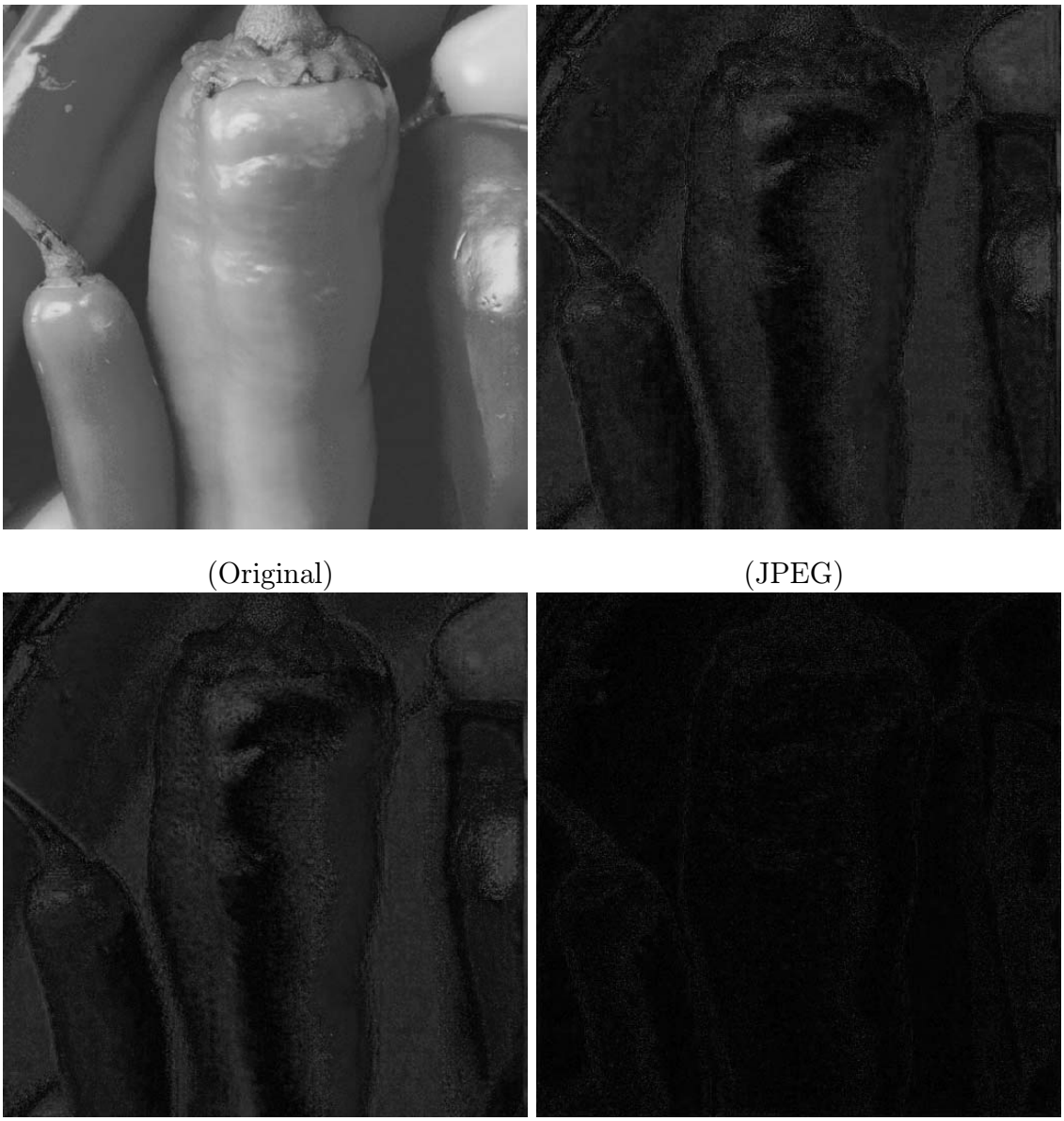
Fig. 5.13. The original and the difference images of *Lenna* at 0.5 bpp using *CEZW*, JPEG and SPIHT. The difference images are enhanced by a factor of 3.64 to show the coding artifacts.



(Original) (JPEG)

(SPIHT) (*CEZW*)

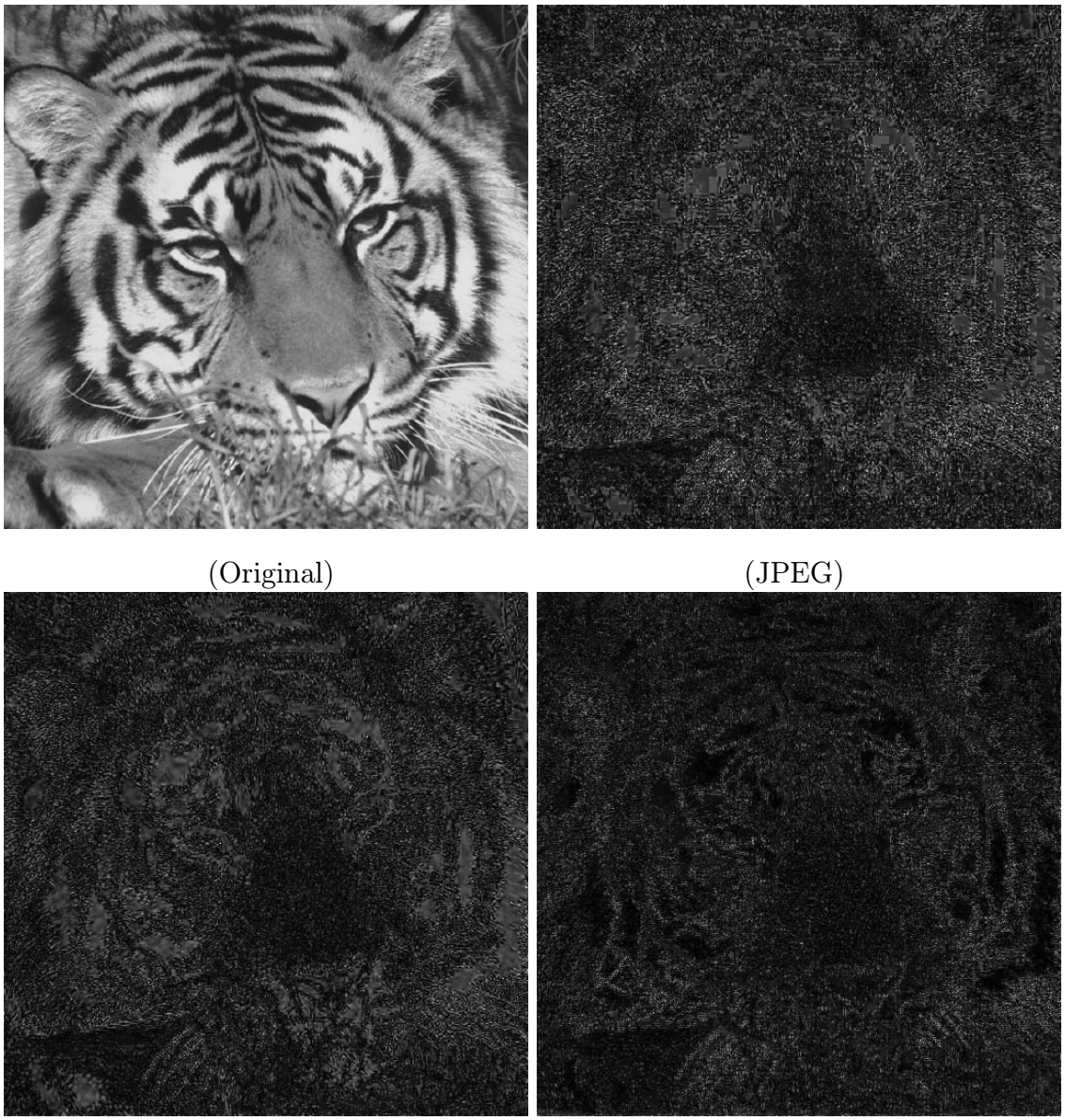
Fig. 5.14. The original and the difference images of *Model* at 0.5 bpp using *CEZW*, JPEG and SPIHT. The difference images are enhanced by a factor of 3.64 to show the coding artifacts.



(Original) (JPEG)

(SPIHT) (*CEZW*)

Fig. 5.15. The original and the difference images of *Peppers* at 0.5 bpp using *CEZW*, JPEG and SPIHT. The difference images are enhanced by a factor of 3.64 to show the coding artifacts.



(Original) (JPEG)

(SPIHT) (*CEZW*)

Fig. 5.16. The original and the difference images of *Tiger* at 0.5 bpp using *CEZW*, JPEG and SPIHT. The difference images are enhanced by a factor of 3.64 to show the coding artifacts.

6. RATE SCALABLE VIDEO CODING

Recently, scalable video compression algorithms have become popular. These algorithms have applications in digital libraries, delivery of video over computer networks, video telephony and multicast of regular resolution TV and high definition TV (HDTV). The MPEG2 video compression standard incorporated several scalable modes, including signal-to-noise ratio (SNR) scalability, spatial scalability and temporal scalability [7, 21]. However, these modes are layered instead of continuously scalable.

For video compression, the embedded coding scheme can be more complicated when compared with progressive image compression since a video sequence contains multiple images. Instead of sending the beginning portion of the bit stream to the decoder, the sender need to selectively provide the decoder with portions of the bit stream corresponding to different frames or sections of frames of the video sequence. These selected portions of the compressed data meet the data rate requirement and can be decoded by the decoder. This approach can be achieved if the position of the bits corresponding to each frame or each section of frames can be identified.

One could achieve continuous rate scalability for a video coder by using a rate scalable still image compression algorithms such as [76, 22, 79] for each video frame. This is known as the "intracoded frames (I frames) only" approach. We applied Shapiro's algorithm [22] separately to 3 color components of each video frame of the *football* sequence. The rate-distortion performance is shown in Figure 6.1. A visually acceptable decoded sequence is available only when the data rate is larger than 2.5 Mb/s for a CIF (352x240) sequence. This low performance is due to the fact that the temporal redundancy in a video sequence is not exploited. Taubman and

Zakhor proposed a embedded scalable video compression algorithm using 3-D subband coding [83]. Some draw backs of their scheme are that the 3-D subband algorithm can not exploit the temporal correlation of the video sequence very efficiently, especially when there is a great deal of motion. Also since 3-D subband decomposition requires multiple frames to be processed at the same time, it requires more memory for both the encoder and the decoder, and results in a longer delay than motion compensated hybrid video compression algorithms such as MPEG and H.263 [6, 7, 9].

Motion compensation is very effective in reducing the temporal redundancy and is commonly used in video coding. A motion compensated hybrid video compression algorithm usually consists of two major parts, the generation and compression of the motion vector (MV) fields and the compression of the I frames and prediction error frames. Motion compensation is usually block based, i.e. the current image is divided into blocks and each block is matched with the reference frame. The best matched block of pixels from the reference frame are then used in the current block. The prediction error frame is obtained by taking the difference between the current frame and the motion predicted frame. PEFs are usually encoded using either block-based transforms, such as DCT [78], or non-block-based coding, such as subband coding or the wavelet transform. DCT is used in MPEG and H.263 algorithms [6, 7, 9]. A major problem with a block-based transform coding algorithm is the existence of the visually unpleasant block artifacts, especially at low data rates. This problem can be eliminated by using the wavelet transform, which is usually obtained over the entire image. The wavelet transform has been used in video coding for the compression of motion predicted error frames [95, 96]. However these algorithms are not scalable. If we use wavelet based rate scalable algorithms to compress the I frames and PEFs, rate scalable video compression can be achieved. Recently, a wavelet based rate scalable video coding algorithm has been proposed by Wang and Ghanbari [93]. In their scheme the motion compensation was done in the wavelet transform domain. However, in the wavelet transform domain spatial shifting results in phase shifting, hence motion compensation does not work well and may cause motion tracking errors

in high frequency bands. In this chapter we propose a new continuous rate scalable hybrid video compression algorithm, the *Scalable Adaptive Motion Compensated Wavelet (SAMCoW)* algorithm. In the *SAMCoW* algorithm, motion compensation is done in the pixel domain.

6.1 Adaptive Motion Compensation (AMC)

One of the problems of any rate scalable compression algorithm is the ability to maintain a constant visual quality at any data rate. Often the distortion of a decoded video sequence varies from frame to frame. Since a video sequence is usually decoded at 25 or 30 frames per second (or 5-15 frames per second for low data rate applications), due to the temporal masking effect, the distortion of each frame may not be discerned as accurately as when individual frames are examined. Yet, the distortion of each frame contributes to the overall perception of the video sequence. When the quality of successive frames decreases for a relatively long time, a viewer will notice the change. This increase in distortion may be visually perceived as an increase in fuzziness and/or blockiness. This phenomenon can occur due to error propagation, which is very common when motion compensated prediction is used. This can be even more serious when using a rate scalable codec.

Motion vector fields are generated by matching the current frame with its reference frame. After the motion vector field is obtained for the current frame, the predicted frame is generated by rearranging the pixels in the reference frame relative to m. We denote this option by $M(\cdot)$, or

$$\boldsymbol{p}_{pred} = M(\boldsymbol{p}_{ref}, \boldsymbol{m}).$$

The prediction error frame is obtained by taking the difference between the current frame and the predicted frame

$$\boldsymbol{p}_{diff} = \boldsymbol{p} - \boldsymbol{p}_{pred}.$$

At the decoder, the predicted frame is obtained by applying the decoded motion vector field to the reference frame at the decoder

$$\hat{\boldsymbol{p}}_{pred} = M(\hat{\boldsymbol{p}}_{ref}, \hat{\boldsymbol{m}}).$$

The decoded frame is then obtained by adding the $\hat{\boldsymbol{p}}_{pred}$ to the decoded PEF $\hat{\boldsymbol{p}}_{diff}$

$$\hat{\boldsymbol{p}} = \hat{\boldsymbol{p}}_{pred} + \hat{\boldsymbol{p}}_{diff}.$$

Since the motion field is losslessly decoded, by maintaining the reference frame at the encoder to be identical to that at the decoder, i.e. $\boldsymbol{p}_{ref} = \hat{\boldsymbol{p}}_{ref}$, then

$$\hat{\boldsymbol{p}}_{pred} = \boldsymbol{p}_{pred}.$$

This results in the decoded PEF, $\hat{\boldsymbol{p}}_{diff}$, being the only source of distortion in $D(\boldsymbol{p}-\hat{\boldsymbol{p}})$. Thus, one can achieve better performance if the encoder and decoder use the same reference frame. For a fixed rate codec $\hat{\boldsymbol{p}}_{ref}$ is the previous decoded frame, this is usually achieved by using a prediction feedback loop in the encoder so that the decoded frames are used as reference frames (Figure 6.2). This procedure is commonly used in MPEG or H.263. However, in our scalable codec, the decoded frames have different distortions at different data rates. Hence, it is impossible for the encoder to generate the exact reference frames as in the decoder for all the possible data rates. One solution is to have the encoder lock on to a fixed data rate (usually the highest data rate) and let the decoder run freely, as in Figure 6.2. The codec will work exactly as the non-scalable codec, when decoding at the highest data rate. However, when the decoder is decoding at a low data rate, the quality of the decoded reference frames at the decoder will deviate from that at the encoder. Hence, both the motion prediction and the decoding of the PEFs contribute to the increase in distortion of the decoded video sequence. This distortion also propagates from one frame to the next within a group of pictures (GOP). If the size of a GOP is large, the increase in distortion can be unacceptable.

To maintain video quality, we need to keep the reference frames the same at both the encoder and the decoder. This can be achieved by adding a feedback loop in the

decoder (Figure 6.3), such that the decoded reference frames at both the encoder and decoder are locked to the same data rate—the lowest data rate. We denote this scheme as *adaptive motion compensation (AMC)* [97, 98]. We assume that the target data rate R is within the range $R_L \leq R \leq R_H$ and the bits required to encode the motion vector fields have data rate R_{MV}, where $R_{MV} < R_L$. At the encoder, since R_{MV} is known, the embedded bit stream can always be decoded at rate $R_\Delta = R_L - R_{MV}$, which is then added to the predicted frame to generate the reference frame $\hat{\boldsymbol{p}}_{ref}$ for the next frame. At the decoder, the embedded bit stream is decoded at two data rates, the targeted data rate $R - R_{MV}$ and the fixed data rate $R_L - R_{MV}$. The frame decoded at rate $R_\Delta = R_L - R_{MV}$ is added to the predicted frame to generate the reference frame, which is exactly the same as the reference frame $\hat{\boldsymbol{p}}_{ref}$ used in the encoder. The frame decoded at rate $R - R_{MV}$ is added to the predicted frame to generate the final decoded frame. This way, the reference frames at the encoder and the decoder are kept identical, which leaves the decoded PEF $\hat{\boldsymbol{p}}_{diff}$ to be the only source of distortion. Hence, error propagation is eliminated.

6.2 Implementation of SAMCoW

We combine the *CEZW* and the *AMC* schemes to establish our new rate scalable video compression algorithm, which is known as *Scalable Adaptive Motion Compensation Wavelet (SAMCoW)*. The discrete wavelet transform was implemented using the biorthogonal wavelet bases from [94] — the 9-7 tap filter banks in particular. Four to six levels of wavelet decomposition were used, depending on the image size.

The video sequences used in our experiments use the YUV color space with color components downsampled to 4:2:0. Motion compensation is implemented using macroblocks, i.e. 16x16 for the Y component and 8x8 for the U and V components, respectively. The search range is ± 15 luminance pixels in both the horizontal and vertical directions. Motion vectors are restricted to integer precision. The spatially corresponding blocks in Y, U and V components share the same motion vector. One problem with block based motion compensation is that it introduces blockiness to

the prediction error images. The blocking edges cannot be efficiently coded using the wavelet transform and may introduce unpleasant ringing effects. To reduce the blockiness in the prediction error images, overlapped block motion compensation is used for the Y component [99, 95, 9]. Let $L^{i,j}$ be the ith row and jth column macroblock of the luminance image and $m^{i,j} = [m_x^{i,j}, m_y^{i,j}]$ be its motion vector. The predicted pixel values for $L^{i,j}$ are the weighted sum

$$\begin{aligned} L^{i,j}(k,l) = \quad & w_c(k,l)L_{ref}^{i,j}(k+m_y^{i,j}, l+m_x^{i,j}) \\ & +w_t(k,l)L_{ref}^{i,j}(k+m_y^{i-1,j}, l+m_x^{i-1,j}) \\ & +w_b(k,l)L_{ref}^{i,j}(k+m_y^{i+1,j}, l+m_x^{i+1,j}) \\ & +w_l(k,l)L_{ref}^{i,j}(k+m_y^{i,j-1}, l+m_x^{i,j-1}) \\ & +w_r(k,l)L_{ref}^{i,j}(k+m_y^{i,j+1}, l+m_x^{i,j+1}), \end{aligned}$$

where $k, l \in \{0 \ldots 15\}$. The weighting values for the current block are

$$w_c = \begin{pmatrix} 4 & 5 & 5 & 5 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 5 & 5 & 5 & 4 \\ 5 & 5 & 5 & 5 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 5 & 5 & 5 & 5 \\ 5 & 5 & 6 & 6 & 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 & 6 & 6 & 5 & 5 \\ 5 & 5 & 6 & 6 & 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 & 6 & 6 & 5 & 5 \\ 6 & 6 & 7 & 7 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 7 & 7 & 6 & 6 \\ 6 & 6 & 7 & 7 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 7 & 7 & 6 & 6 \\ 6 & 6 & 7 & 7 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 7 & 7 & 6 & 6 \\ 6 & 6 & 7 & 7 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 7 & 7 & 6 & 6 \\ 6 & 6 & 7 & 7 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 7 & 7 & 6 & 6 \\ 6 & 6 & 7 & 7 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 7 & 7 & 6 & 6 \\ 6 & 6 & 7 & 7 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 7 & 7 & 6 & 6 \\ 6 & 6 & 7 & 7 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 7 & 7 & 6 & 6 \\ 5 & 5 & 6 & 6 & 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 & 6 & 6 & 5 & 5 \\ 5 & 5 & 6 & 6 & 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 & 6 & 6 & 5 & 5 \\ 5 & 5 & 5 & 5 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 5 & 5 & 5 & 5 \\ 4 & 5 & 5 & 5 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 5 & 5 & 5 & 4 \end{pmatrix} /8.$$

The weighting values for the top block are

$$w_t = \begin{pmatrix} 2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2 \\ 1&1&2&2&2&2&2&2&2&2&2&2&2&2&1&1 \\ 1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1 \\ 1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \end{pmatrix} /8,$$

and the weighting values for the left block are

$$w_l = \begin{pmatrix} 2&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0 \\ 2&2&1&1&0&0&0&0&0&0&0&0&0&0&0&0 \\ 2&2&1&1&0&0&0&0&0&0&0&0&0&0&0&0 \\ 2&2&1&1&0&0&0&0&0&0&0&0&0&0&0&0 \\ 2&2&1&1&0&0&0&0&0&0&0&0&0&0&0&0 \\ 2&2&1&1&0&0&0&0&0&0&0&0&0&0&0&0 \\ 2&2&1&1&0&0&0&0&0&0&0&0&0&0&0&0 \\ 2&2&1&1&0&0&0&0&0&0&0&0&0&0&0&0 \\ 2&2&1&1&0&0&0&0&0&0&0&0&0&0&0&0 \\ 2&2&1&1&0&0&0&0&0&0&0&0&0&0&0&0 \\ 2&2&1&1&0&0&0&0&0&0&0&0&0&0&0&0 \\ 2&2&1&1&0&0&0&0&0&0&0&0&0&0&0&0 \\ 2&2&1&1&0&0&0&0&0&0&0&0&0&0&0&0 \\ 2&2&1&1&0&0&0&0&0&0&0&0&0&0&0&0 \\ 2&2&1&1&0&0&0&0&0&0&0&0&0&0&0&0 \\ 2&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0 \end{pmatrix} /8$$

The weighting values for the bottom and right blocks are $w_b(i,j) = w_t(15-i,j)$ and $w_r(i,j) = w_l(i,15-j)$, respectively, where $i,j \in \{0\ldots 15\}$. Obviously, $w_t(i,j) + w_b(i,j) + w_l(i,j) + w_r(i,j) = 1$, which is the necessary condition for overlapped motion compensation. The motion vectors are differentially coded. The prediction of the motion vector for the current macroblock is obtained by taking the median

of the motion vectors of the left, the top and the top-right adjacent macroblocks. The difference between the current motion vector and the predicted motion vector is entropy coded.

In our experiments, the GOP size is 100 or 150 frames and the first frame of a GOP is an intra-coded frame. To maintain the video quality of a GOP, the intra-coded frames need to be encoded with relatively more bits. We encode an intra-coded frame using 6 to 10 times the number of bits used for each predictively coded frame. In our experiments, no bidirectionally predictive-coded frames (B frames) are used. However, the nature of our algorithm does not preclude the use of B frames.

The embedded bit stream is arranged as follows. The necessary header information, such as the resolution of the sequence and the number of levels of the wavelet transform, is embedded at the beginning of the sequence. In each GOP, the I-frame is coded first using our rate scalable coder. For each P-frame, the motion vectors are differentially coded first. The PEF is then compressed using our rate scalable algorithm. When decoding, after sending the bits of each frame, an end-of-frame (EOF) symbol is transmitted. The decoder can then decode the sequence without prior knowledge of the data rate. Therefore the data rate can be changed dynamically in the process of decoding.

6.3 Experimental Results and Discussion

We use the term "visual quality" of a video sequence (or an image) to describe the fidelity, or the closeness, of the decoded to the original video sequence (or image) when perceived by a viewer. We are not aware of a computable metric that will accurately predict how a human observer will perceive a decompressed video sequence. In this paper we will use the peak signal-to-noise ratio (PSNR) based on mean-square error as our "quality" measure. We feel this measure, while unsatisfactory, does track "quality" in some sense. PSNR of the color component X, $X \in \{Y, U, V\}$, is obtained

by:

$$SNR_X = 10 \log \frac{255^2}{mse(X)},$$

where $mse(X)$ is the mean square error of X. When necessary, the overall or combined PSNR is obtained by:

$$SNR = 10 \log \frac{255^2}{(mse(Y) + mse(U) + mse(V))/3}.$$

The effectiveness of using *AMC* is shown in Figure 6.4. From the figure we can see that the non-*AMC* algorithm works better at the highest data rate, to which the encoder feedback loop is locked. However, for any other data rates, the PSNR performance of the non-*AMC* algorithm declines very rapidly while the error propagation is eliminated in the *AMC* algorithm. Data rate scalability can be achieved and video quality can be kept relatively constant even at a low data rate with *AMC*. One should note that the *AMC* scheme can be incorporated into any motion compensated rate scalable algorithm, no matter what kind of transform is used for the encoding of I frames and PEFs.

In our experiment, two sets of video sequences are used. One set is CIF (352x240) sequences with 30 frames per second. The other set is QCIF (176x144) sequences with 10 frames per second or 15 frames per second [1].

The CIF sequences are decompressed using *SAMCoW* at data rates of 1 megabits per second (Mb/s), 1.5 Mb/s, 2 Mb/s, 4 Mb/s and 6 Mb/s. The representative frames decoded at the above rates is shown in Figures 6.5 and 6.6. At 6 Mb/s, the distortion is imperceptible. The decoded video has an acceptable quality when the data rate is 1 Mb/s. We used Taubman and Zakhor's algorithm [83] and MPEG-1 to encode/decode the same sequences at the above data rates [2]. Since MPEG-1 is not scalable, the sequences were specifically compressed and decompressed at each of the above data rates. The overall PSNRs of each frame in a GOP are shown in Figures 6.7 and 6.8. The computational rate-distortion in terms of average PSNR

[1] The original sequences along with the decoded sequences using *SAMCoW* are available at `ftp://skynet.ecn.purdue.edu/pub/dist/delp/samcow`.

[2] Taubman and Zakhor's software was obtained from the authors.

over a GOP is shown in Table 6.1. The data indicates that *SAMCoW* has very comparable performance to the other methods tested. Comparison of the decoded image quality using *SAMCoW*, Taubman and Zakhor's algorithm and MPEG-1 is shown in Figures 6.9 and 6.10. We can see that *SAMCoW* out performs Taubman and Zakhor's algorithm, visually and in terms of PSNR. Even though *SAMCoW* does not perform as well as MPEG-1 in terms of PSNR, subjective experiments have shown that our algorithm produces decoded video with comparable visual quality as MPEG-1 at every tested data rate.

The QCIF sequences are compressed and decompressed using *SAMCoW* at data rates of 20 kilobits per second (Kb/s), 32 Kb/s, 64 Kb/s, 128 Kb/s, and 256 Kb/s. The same set of sequences are compressed using the H.263 algorithm at the above data rates [3]. Again, since H.263 is not scalable, the sequences were specifically compressed and decompressed at each of the above data rates. Decoded images using *SAMCoW* at different data rates, along with that using H.263, are shown in Figures 6.11, 6.12 and 6.13. The overall PSNRs of each frame in a GOP are shown in Figures 6.14 and 6.15. The computational rate-distortion in terms of average PSNR over a GOP is shown in Tables 6.2 and 6.3. Our subjective experiments have shown that at data rates greater than 32 Kb/s *SAMCoW* performs similar to H.263. Below 32 Kb/s when sequences with high motion are used, such as the *Foreman* sequence, our algorithm is visually slightly inferior to H.263. This is partially due to the fact that the wavelet transform is obtained for the entire image and the algorithm cannot allocate extra bits to areas with high activity. It should be emphasized that the scalable nature of *SAMCoW* makes it very attractive in many low bit rate applications, e.g. streaming video on the Internet. Furthermore, the decoding data rate can be dynamically changed.

[3] The H.263 software was obtained from `ftp://bonde.nta.no/pub/tmn/software`.

Table 6.1 PSNR of CIF sequences, average over a GOP. (30 frames per second)

sequence		*football*				*flowergarden*			
components		All	Y	U	V	All	Y	U	V
1 Mb/s	*SAMCoW*	27.1	25.8	26.6	29.9	24.3	22.4	24.4	27.3
	Taubman	25.2	21.5	28.8	32.2	21.1	17.3	24.7	28.9
	MPEG-1	28.8	25.7	31.2	33.3	25.7	22.5	27.2	29.9
1.5 Mb/s	*SAMCoW*	28.1	27.1	27.7	30.1	25.4	24.1	25.2	27.8
	Taubman	26.3	22.6	29.6	32.8	21.6	17.9	25.0	29.4
	MPEG-1	30.2	27.3	32.2	33.9	27.0	24.4	28.5	30.4
2 Mb/s	*SAMCoW*	28.8	28.2	28.4	30.4	26.7	25.7	26.6	28.4
	Taubman	26.7	23.1	30.1	33.0	22.3	18.7	25.5	29.6
	MPEG-1	31.2	28.5	33.0	34.5	28.8	26.5	29.9	31.3
4 Mb/s	*SAMCoW*	30.9	30.9	30.4	31.8	28.7	28.8	28.2	29.3
	Taubman	28.9	25.0	31.5	33.9	24.1	20.6	26.7	30.7
	MPEG-1	34.2	32.2	35.3	36.1	32.6	30.9	33.6	34.1
6 Mb/s	*SAMCoW*	34.9	35.3	34.6	35.1	33.8	34.8	33.4	33.4
	Taubman	29.8	26.5	32.6	34.6	25.6	22.3	28.1	31.0
	MPEG-1	36.8	35.2	37.8	38.2	35.4	33.8	36.4	36.5

Table 6.2 PSNR of QCIF sequences, averaged over a GOP. (15 frames per second)

sequence		*akiyo*				*foreman*			
components		All	Y	U	V	All	Y	U	V
20 Kb/s	*SAMCoW*	32.8	31.7	32.3	35.0	28.6	26.1	30.4	31.4
	H.263	37.5	35.6	38.1	40.1	30.3	27.2	33.5	33.6
32 Kb/s	*SAMCoW*	34.6	33.3	34.4	36.8	30.1	27.3	33.4	32.2
	H.263	39.1	37.4	39.7	41.2	31.1	28.1	34.0	34.2
64 Kb/s	*SAMCoW*	38.3	37.2	38.7	39.4	31.6	29.4	33.6	33.4
	H.263	43.1	40.8	44.8	45.1	33.9	31.1	36.2	36.8
128 Kb/s	*SAMCoW*	43.2	41.9	43.7	44.3	33.3	31.5	34.4	34.7
	H.263	46.3	44.6	47.4	47.7	36.6	34.0	38.3	39.3
256 Kb/s	*SAMCoW*	47.5	46.8	47.9	48.1	35.2	34.1	35.5	36.4
	H.263	49.1	48.4	49.3	49.7	38.4	36.1	39.8	41.1

Table 6.3 PSNR of QCIF sequences, averaged over a GOP. (10 frames per second)

sequence		*akiyo*				*foreman*			
components		All	Y	U	V	All	Y	U	V
20 Kb/s	*SAMCoW*	34.0	32.5	33.9	36.3	29.5	26.6	32.7	31.9
	H.263	39.3	36.6	41.4	42.4	30.4	27.1	33.9	34.2
32 Kb/s	*SAMCoW*	36.3	34.9	36.5	38.0	30.5	28.0	33.2	32.3
	H.263	40.3	38.7	40.8	42.1	32.3	29.3	35.2	35.5
64 Kb/s	*SAMCoW*	40.3	38.9	40.8	41.5	32.0	30.0	33.6	33.7
	H.263	43.4	42.1	44.0	44.8	34.8	32.0	37.0	37.7
128 Kb/s	*SAMCoW*	44.9	43.9	45.6	45.7	33.9	32.4	34.8	35.3
	H.263	46.4	44.7	47.5	47.6	37.2	34.7	38.7	39.9
256 Kb/s	*SAMCoW*	48.8	48.6	48.7	49.0	36.1	35.3	36.0	37.3
	H.263	49.3	48.7	49.5	49.9	39.3	37.1	40.5	41.9

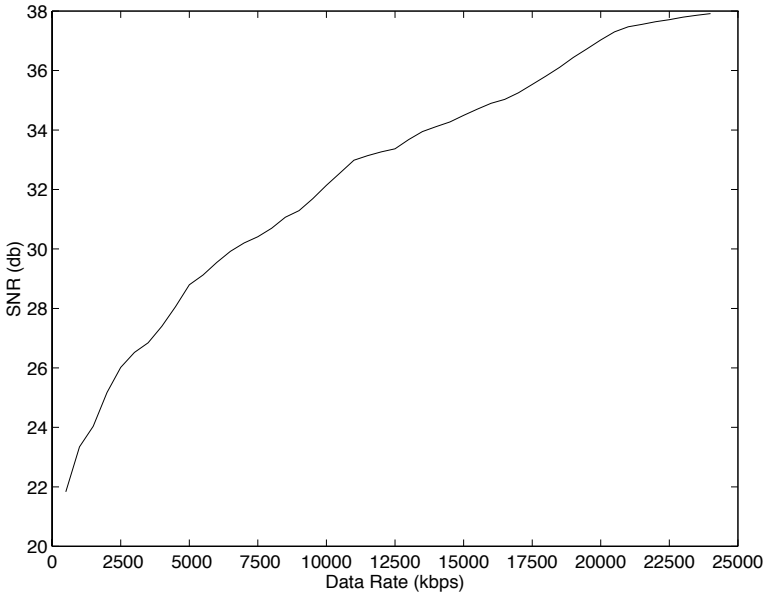


Fig. 6.1. Average PSNR of EZW encoded *football* sequence (I frame only) at different data rates. (30 frames per second)

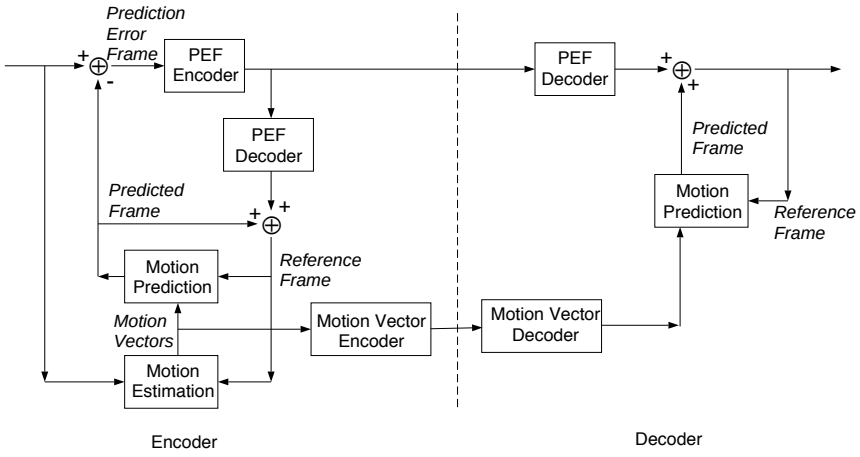


Fig. 6.2. Block diagram of a generalized hybrid video codec for predictively coded frames. Feedback loop is used in the encoder. Adaptive motion compensation is not used.

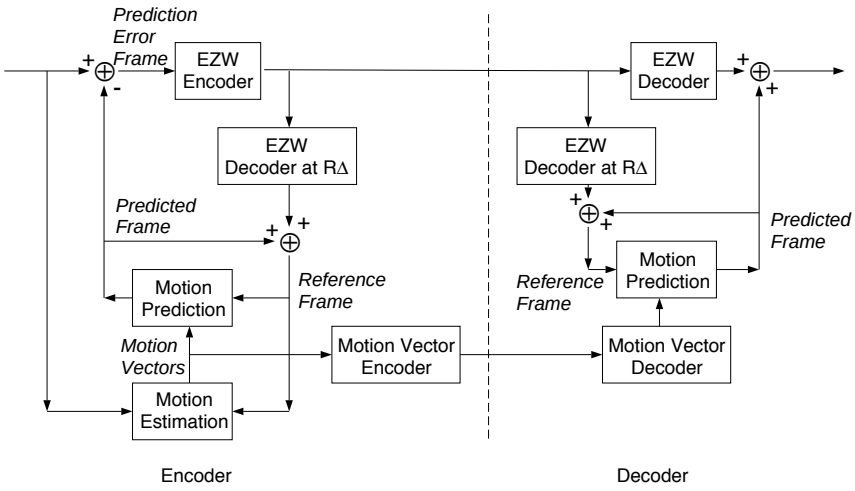


Fig. 6.3. Block diagram of the proposed codec for predictively coded frames. Adaptive motion compensation is used.

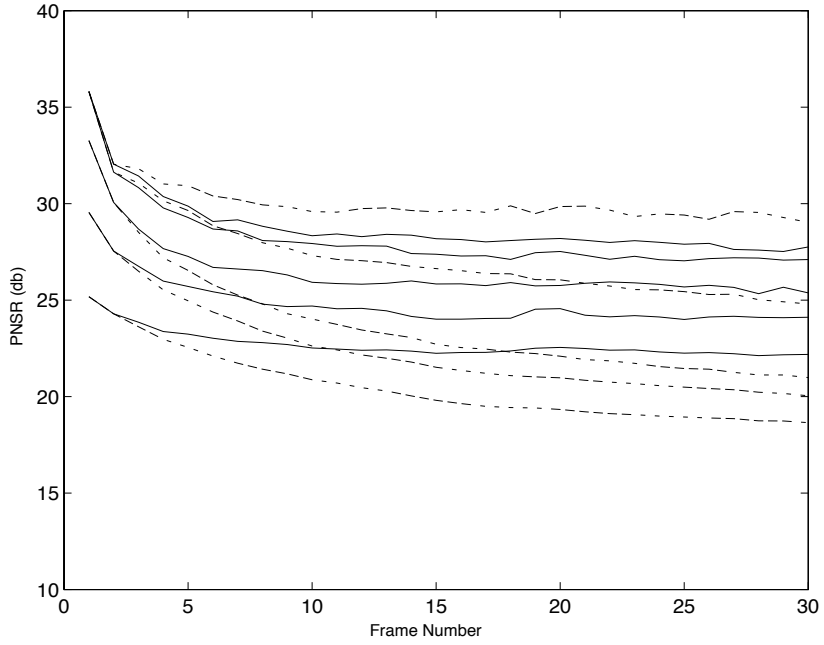
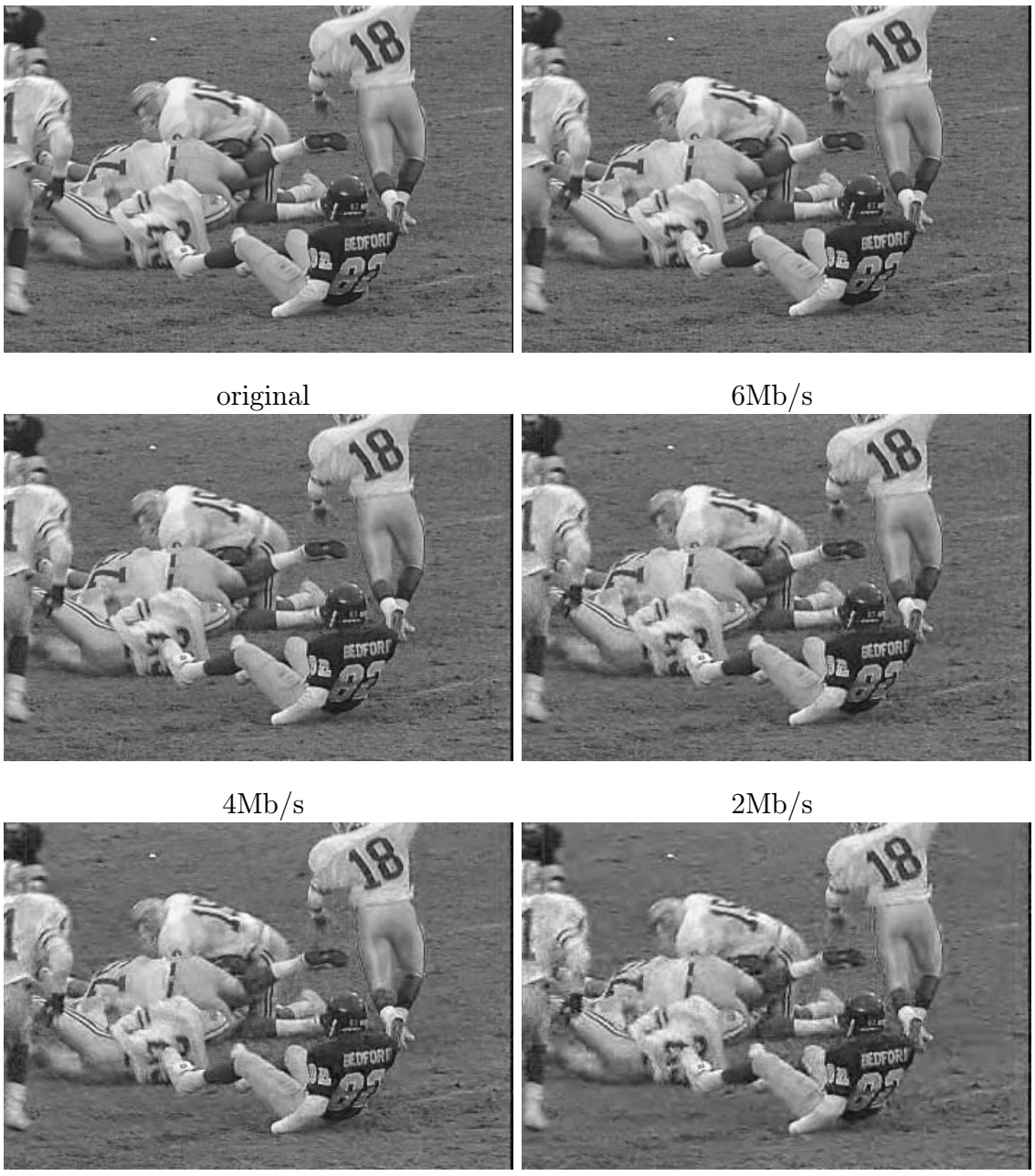


Fig. 6.4. PSNR of each frame within a GOP of the *football* sequence at different data rates. Solid lines: *AMC*; dashed lines: non-*AMC*; Data rates in Kb/s(from top to bottom): 6000, 5000, 3000, 1500, 500.



original 6Mb/s

4Mb/s 2Mb/s

1.5Mb/s 1Mb/s

Fig. 6.5. Frame 35 of the *football* sequence, decoded at different data rates using *SAMCoW* (CIF, 30 frames per second).

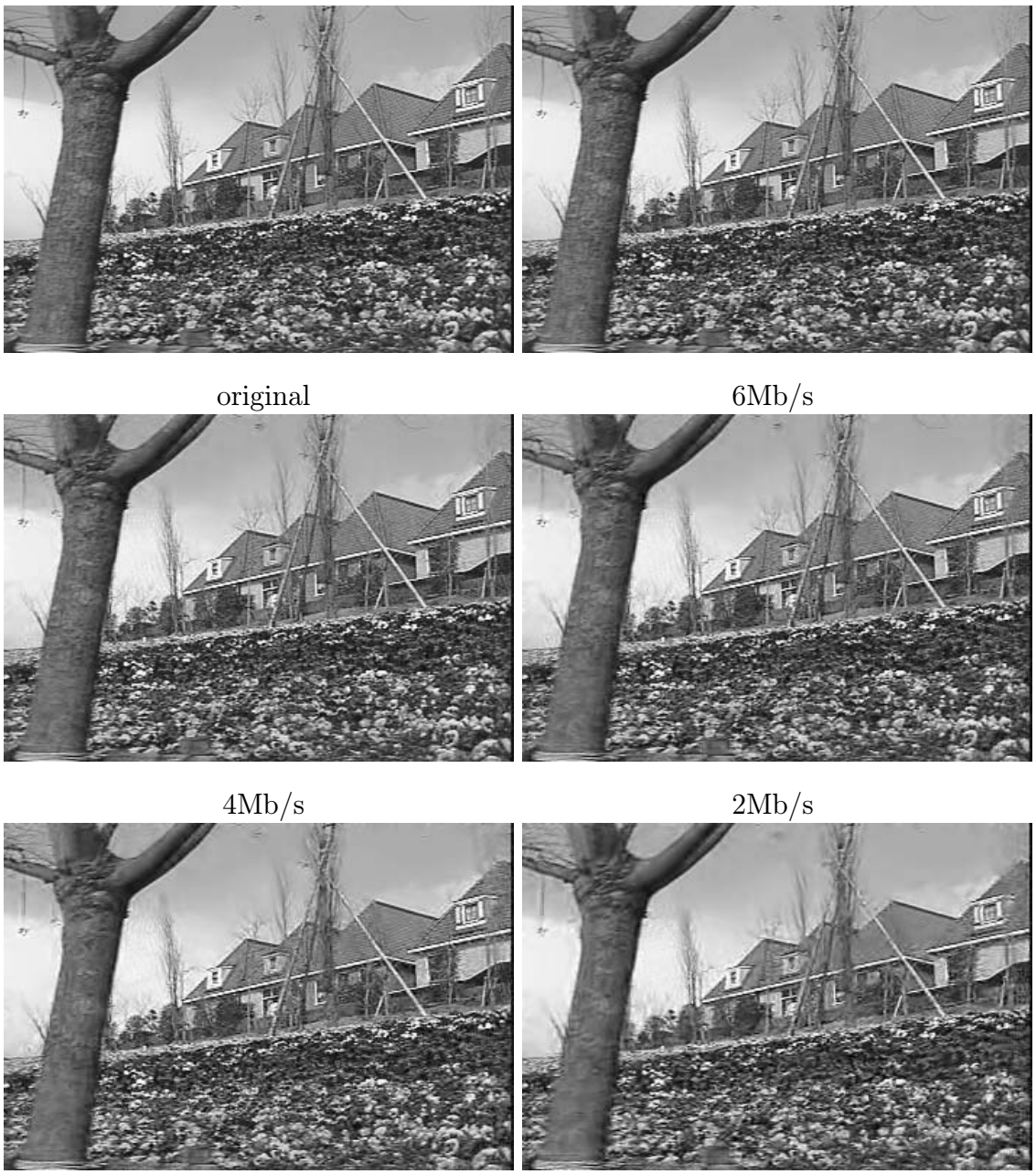
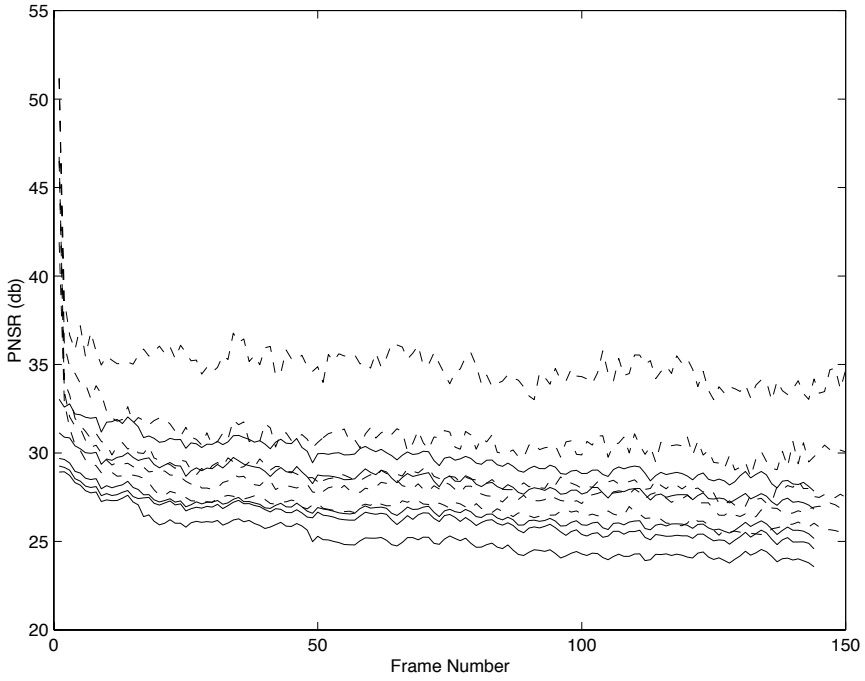
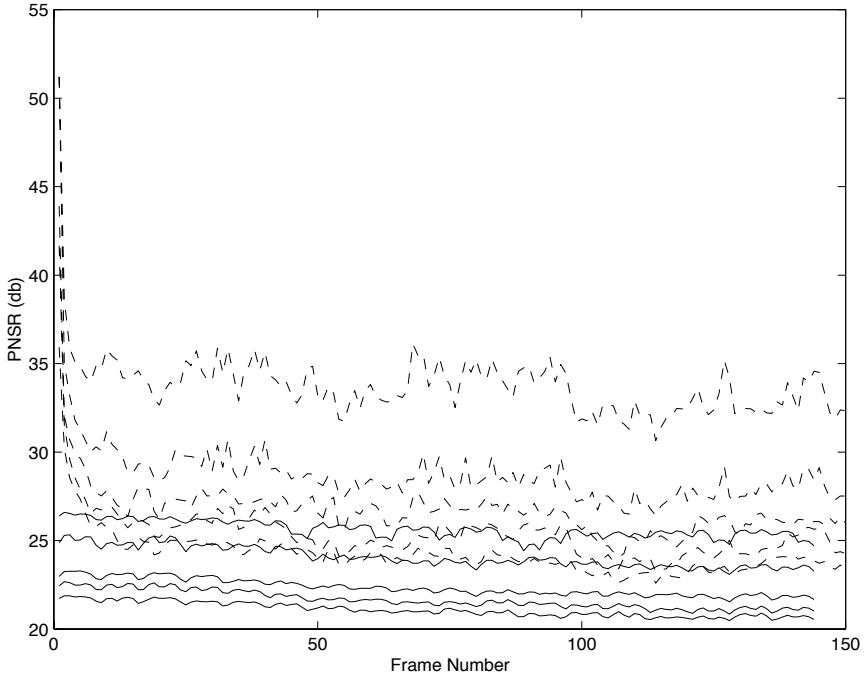


Fig. 6.6. Frame 35 of the *flower* sequence, decoded at different data rates using *SAMCoW* (CIF, 30 frames per second).

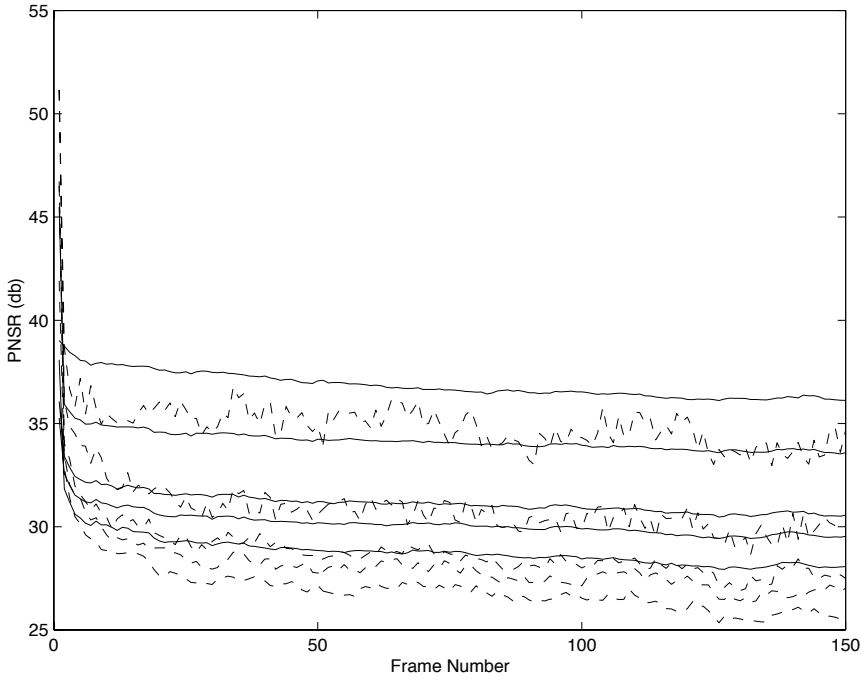


a. *football*

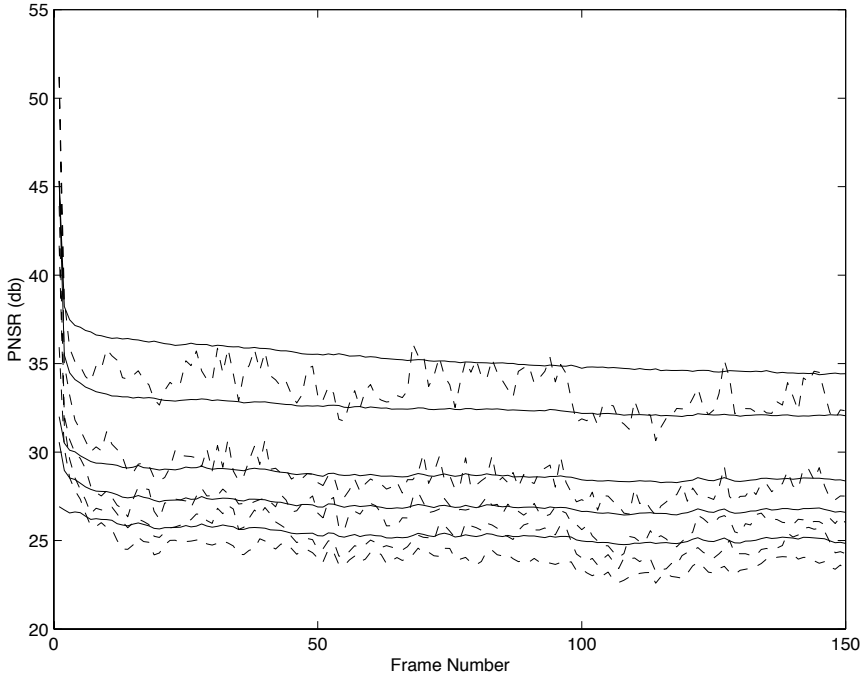


b. *flowergarden*

Fig. 6.7. Comparison of the performance of *SAMCoW* and Taubman and Zakhor's algorithm. Dashed lines: *SAMCoW*; solid lines: Taubman and Zakhor's algorithm. The sequences are decoded at 6 Mb/s, 4 Mb/s, 2 Mb/s, 1.5 Mb/s and 1 Mb/s, which respectively correspond to the lines from top to bottom.

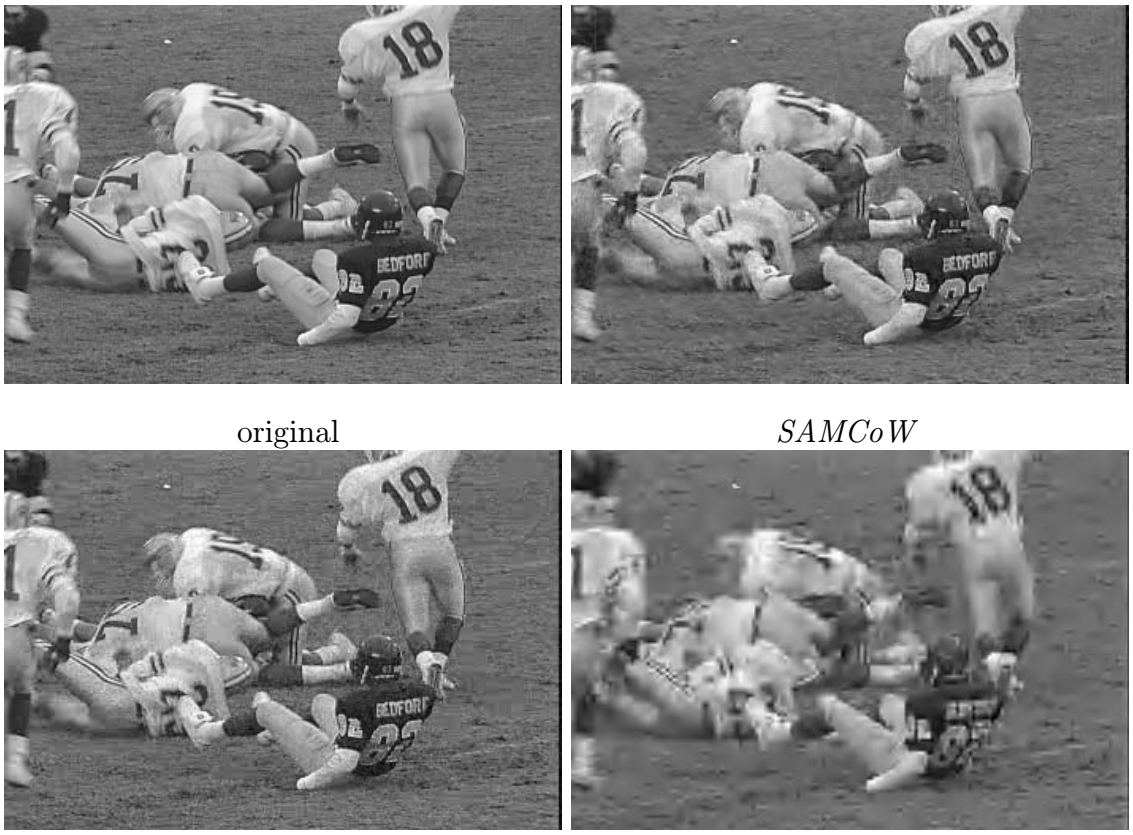


a. *football*



b. *flowergarden*

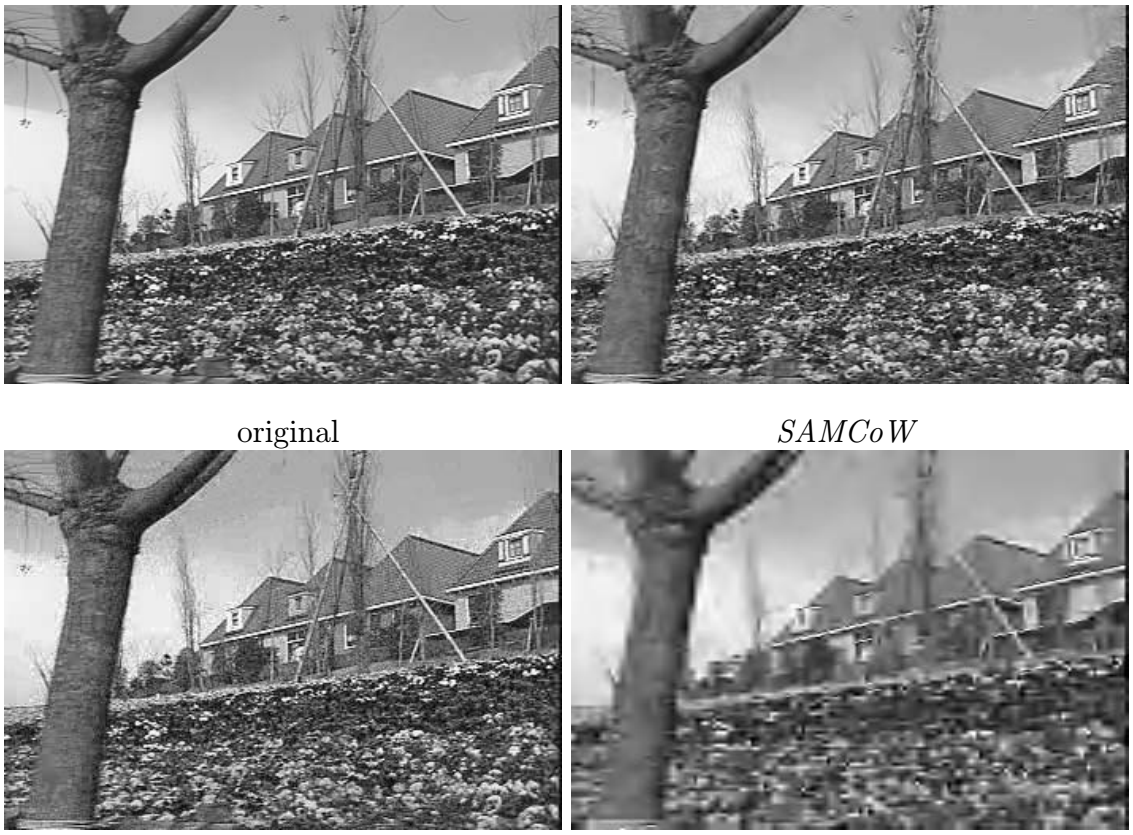
Fig. 6.8. Comparison of the performance of *SAMCoW* and MPEG-1. Dashed lines: *SAMCoW*; solid lines: MPEG-1. The sequences are decoded at 6 Mb/s, 4 Mb/s, 2 Mb/s, 1.5 Mb/s and 1 Mb/s, which respectively correspond to the lines from top to bottom.



original | *SAMCoW*

MPEG-1 | Taubman and Zakhor

Fig. 6.9. Frame 35 of the *football* sequence (CIF, 30 frames per second). The data rate is 1.5 Mb/s



original | *SAMCoW*

MPEG-1 | Taubman and Zakhor

Fig. 6.10. Frame 35 of the *flower* sequence (CIF, 30 frames per second). The data rate is 1.5 Mb/s

Fig. 6.11. Frame 78 of the *Akiyo* sequence (QCIF, 10 frames per second), decoded at different data rates. Left column: *SAMCoW*, right column: H.263.

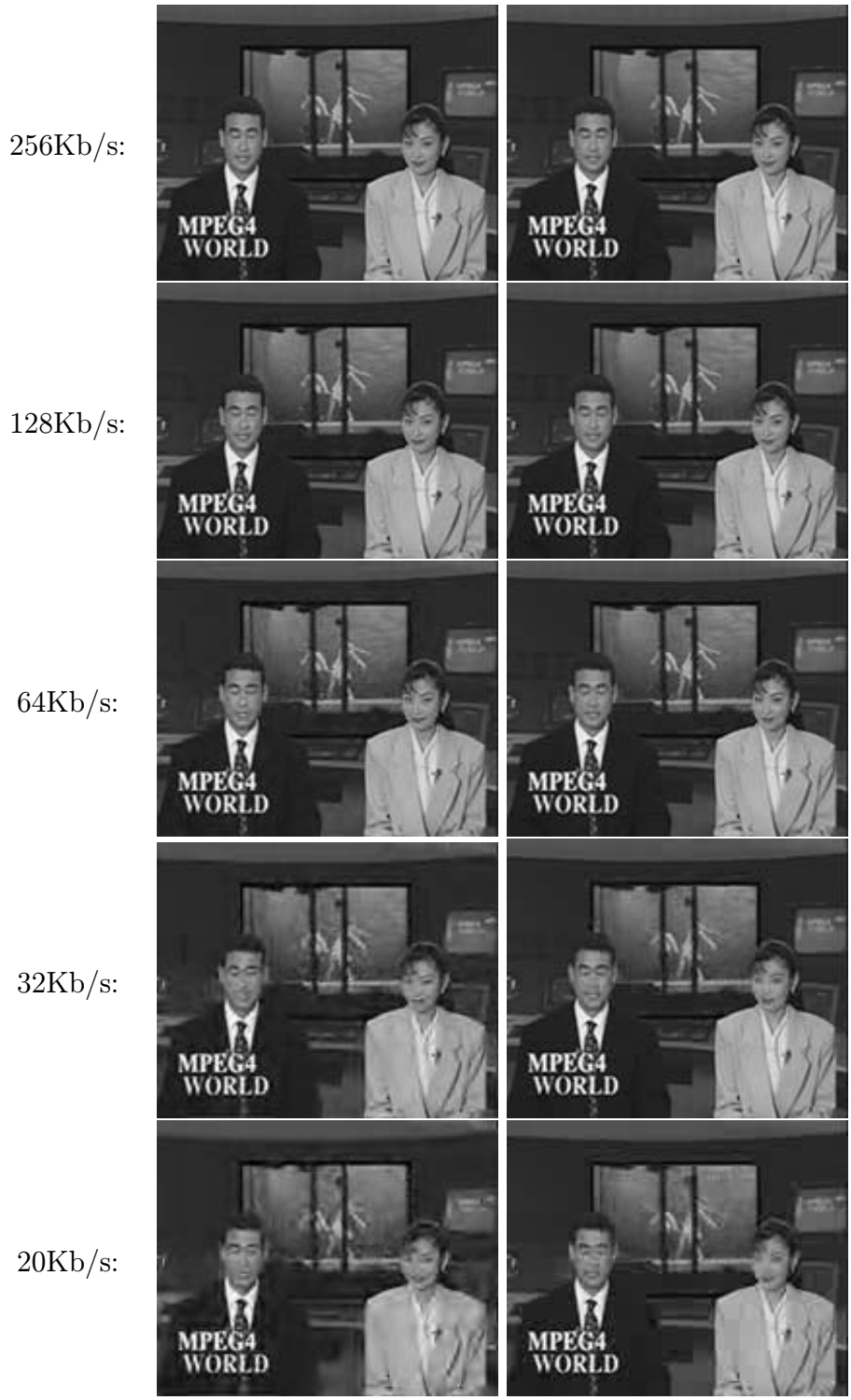
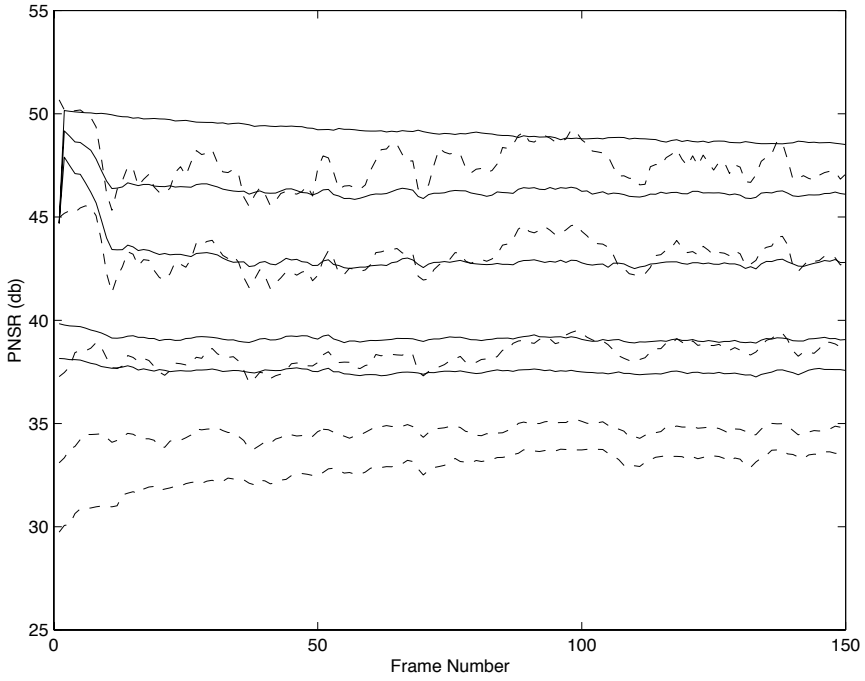
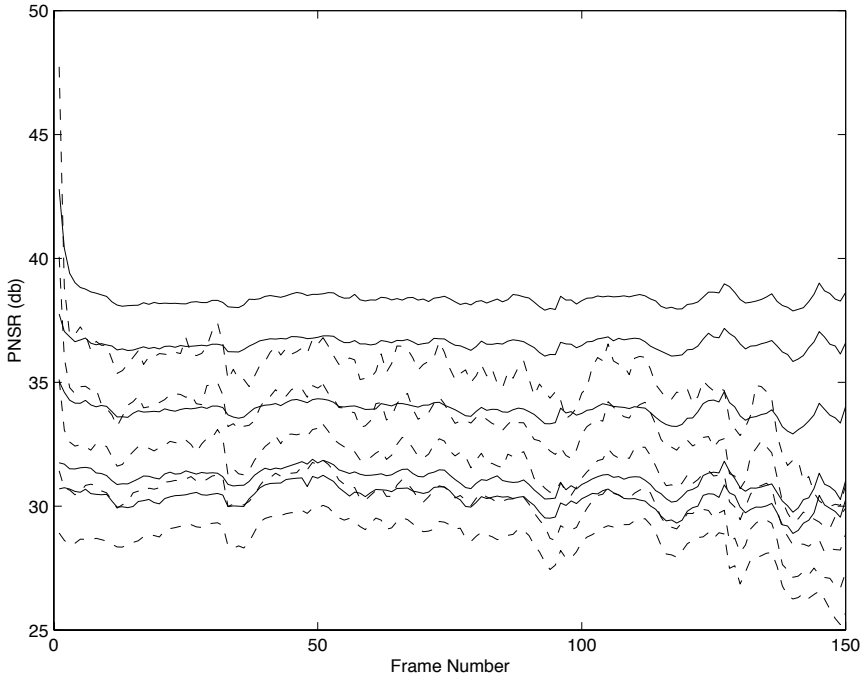


Fig. 6.12. Frame 78 of the *News* sequence (QCIF, 10 frames per second), decoded at different data rates. Left column: *SAMCoW*, right column: H.263.

Fig. 6.13. Frame 35 of the *Foreman* sequence (QCIF, 10 frames per second), decoded at different data rates. Left column: *SAMCoW*, right column: H.263

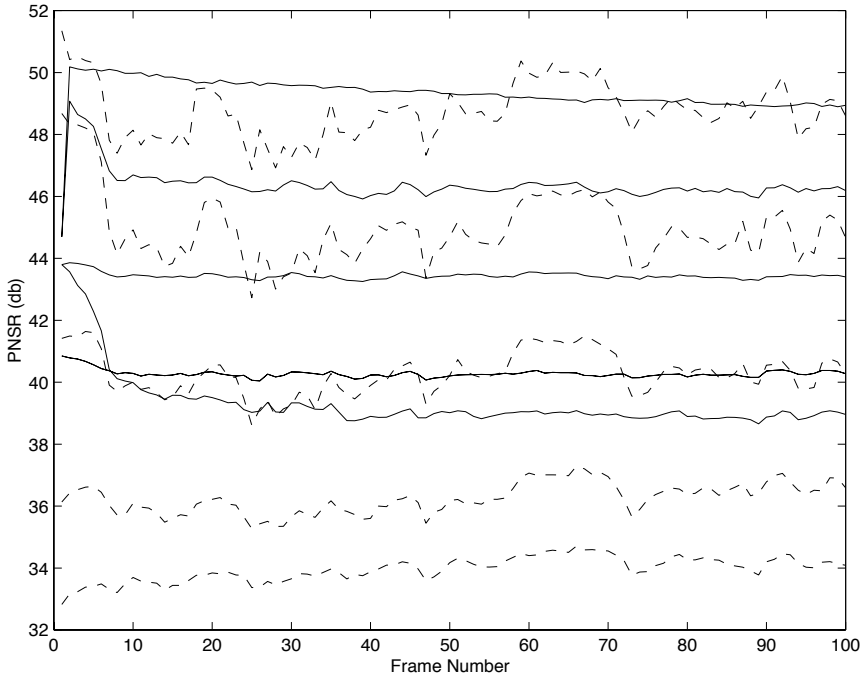


a. *akiyo*

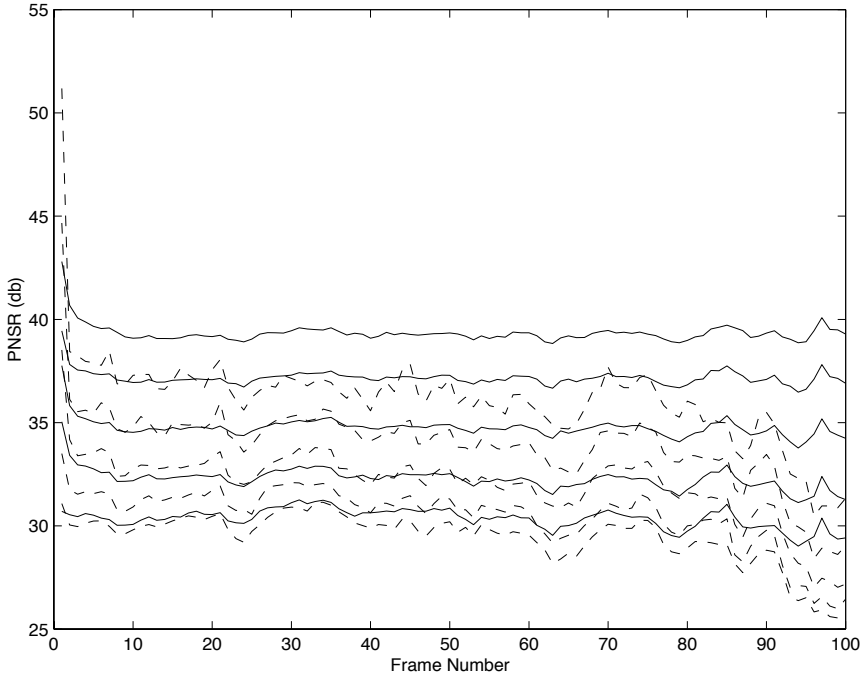


b. *foreman*

Fig. 6.14. Comparison of the performance of *SAMCoW* and H.263. (QCIF at 15 frames per second) Dashed lines: *SAMCoW*; solid lines: H.263. The sequences are decoded at 256 Kb/s, 128 Kb/s, 64 Kb/s, 32 Kb/s and 20 Kb/s, which respectively correspond to the lines from top to bottom.



a. *akiyo*



b. *foreman*

Fig. 6.15. Comparison of the performance of *SAMCoW* and H.263. (QCIF at 10 frames per second) Dashed lines: *SAMCoW*; solid lines: H.263. The sequences are decoded at 256 Kb/s, 128 Kb/s, 64 Kb/s, 32 Kb/s and 20 Kb/s, which respectively correspond to the lines from top to bottom.

7. SUMMARY

7.1 Summary

In this thesis, we addressed two issues related to video compression. One is how to achieve real-time video compression using high performance parallel computers, and the other is rate scalable image and video compression.

For a high performance parallel computer, such as the Intel Paragon, that is optimized for scientific computation, it is difficult to maintain a huge data throughput which is required for real-time video compression. While the Paragon has the computation capacity to compress video sequences at a speed faster than real-time, we found that real-time performance cannot be achieved if the Input/Output (I/O) techniques are not designed properly.

In this thesis we presented several parallel approaches to the implementation of an MPEG encoder on the Intel Touchstone Delta and the Intel Paragon. In particular, the temporal and spatial-temporal parallel approaches were proposed. We designed the I/O algorithm to reduce I/O contention. We showed that without changing either the hardware configuration or the operating system of a parallel computer, real-time performance in the case of ITU-R 601 video can be achieved using the spatial-temporal approach. We also believe that if the I/O bandwidth of the Touchstone Delta or Paragon can be increased, by changing either the hardware configuration or the operating system, the overall performance can be further improved.

We also studied rate scalable image and video compression. A new wavelet based coding algorithm for color images using a luminance/chrominance color space is presented. Data rate scalability is achieved by using an embedded coding scheme. A

spatial-orientation tree that links not only the frequency bands but also the color components to scan the wavelet coefficients was used. The interdependence between color components is automatically exploited. This technique is known as the *Color Embedded Zerotree Wavelet (CEZW)* algorithm. Based on the CEZW, we further proposed a new continuous rate scalable hybrid video compression algorithm, the *Scalable Adaptive Motion Compensated Wavelet (SAMCoW)* algorithm. *SAMCoW* uses motion compensation to reduce temporal redundancy. The prediction error frames (PEFs) and the intra-coded frames (I frames) are encoded using our new rate scalable color image compression algorithm. An adaptive motion compensation scheme is used, which keeps the reference frames used in motion prediction at both the encoder and decoder identical at any data rate. Thus error propagation can be eliminated, even at a low data rate. Experimental results show that *SAMCoW* has a wide range of rate scalability and achieves comparable performance to the more traditional hybrid video coders, such as MPEG1 and H.263. The nature of *SAMCoW* allows the decoding data rate to be dynamically changed. The data rate can be adjusted to meet network loading, which is very appealing to network oriented applications.

7.2 Future Work

Parallel high performance computers have been mainly used as computational engines for paradigms such as solving partial differential equations or modeling/visualizing DNA structures [18, 19, 20]. Since the many tasks/functions of a video server, besides video compression, are computationally intensive, parallel computers have the potential for being used as platforms for video servers. Some of the functions of a video server that require intensive computation as well as the handling of huge amounts of data include video parsing and video indexing [100]. Others include query handling and video retrieval. Furthermore, a video server must be capable of handling interactive requests from multiple users as well guaranteeing continuous data stream transmission to each user. The use of parallel processing in the implementation of such applications/functions warrants further investigation.

The difference images obtained from motion compensation exhibit different statistical properties as the original images. The difference images usually contain regions with different activities. At regions where motion is well compensated, the energy level is usually very low. At edges of these regions, the difference image usually contains high frequency signals. At other regions where motion compensation can not serve up to its purpose, such as regions that contain newly revealed objects, the behavior of the difference image is similar to that of the original images. An assumption that the use of zerotree is based on is that for the same spatial location and orientation the coefficient with higher frequency usually have lower magnitude. For those regions contain mainly high frequency signals, this assumption may be violated. If regions with different activities can be identified and segmented efficiently, the coding of the difference images may be improved by using different coding strategies for different regions.

Another possible improvement for the *SAMCoW* algorithm is to implement a rate control scheme. Currently, the algorithm assigns same amount of bits for each frame with the same type (I or P). If the bits allocation to each frame can be adaptive, the performance of the algorithm may be improved.

7.3 Publications

Part of the work presented in this thesis has been published (or submitted) in the following papers:

- "An overview of parallel processing approaches to image and video compression," *Proceedings of the SPIE Conference on Image and Video Compression*, 1994 [23].
- "A parallel implementation of an MPEG encoder: Faster than real-time!" *Proceedings of the SPIE Conference on Digital Video Compression: Algorithms and Technologies*, 1995 [13] .

- "A Spatial-Temporal Parallel Approach for Real-Time MPEG Video Compression," *Proceedings of the 25th International Conference on Parallel Processing*, 1996 [31] .

- "Parallel Scalable Libraries and Algorithms for Computer Vision," *Proceedings of the 1994 12th International Conference on Pattern Recognition*, 1994 [101].

- "Parallel Approaches To Real-Time MPEG Video Compression," submitted to *Journal of Parallel and Distributed Computing* [102].

- "Rate-scalable Video Coding Using a Zerotree Wavelet Approach," *Proceedings of the Ninth IEEE Image and Multidimensional Digital Signal Processing Workshop*, 1996 [97].

- "A Control Scheme for a Data Rate Scalable Video Codec," *Proceedings of the IEEE International Conference on Image Processing*, 1996 [98].

- "Wavelet Based Rate Scalable Video Compression," submitted to the *IEEE Transactions on Circuits and Systems for Video Technology* [103].

- "A fast algorithm for video parsing using mpeg compressed sequences," *Proceedings of the IEEE International Conference on Image Processin g*, 1995 [100].

REFERENCES

[1] S. W. Smoliar and H. Zhang, "Content-based video indexing and retrieval," *IEEE Multimedia*, pp. 62–72, Summer 1994.

[2] CCIR, *Recommendation 601-2: Encoding parameters of digital television for studio*. International Consultative Committee for Radio, 1990.

[3] A. M. Tekalp, *Digital video processing*. New Jersey: Prentice Hall PTR, 1995.

[4] R. Hopkins, "Digital terrestrial HDTV for North America: the Grand Alliance HDTV System," *IEEE Transactions on Consumer Electronics*, vol. 40, no. 3, pp. 185–198, August 1994.

[5] CCITT, *Recommendation H.261: Video codec for audiovisual services at p*64 kbit/s*. The International Telegraph and Telephone Consultative Committee, 1990.

[6] ISO/IEC 11172-2, *Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbit/s*. MPEG (Moving Pictures Expert Group), International Organization for Standardization, 1993. (MPEG1 Video).

[7] ISO/IEC 13818-2, *Generic coding of moving pictures and associated audio information*. MPEG (Moving Pictures Expert Group), International Organization for Standardization, 1994. (MPEG2 Video).

[8] K. R. Rao and J. J. Hwang, *Techniques and standards for image, video and audio coding*. New Jersey: Prentice Hall PTR, 1996.

[9] ITU-T, *ITU-T Recommendation H.263: Video coding for low bitrate communication*. The International Telecommunication Union, 1996.

[10] J. R. Jain and A. K. Jain, "Displacement measurement and its application in interframe image coding," *IEEE Transactions on Communications*, vol. COM-29, no. 12, pp. 1799–1808, December 1981.

[11] D. Le Gall, "MPEG: A video compression standard for multimedia applications," *Communications of the ACM*, vol. 34, no. 4, pp. 46–58, April 1991.

[12] H. G. Musmann, P. Pirsch, and H.-J. Grallert, "Advances in picture coding," *Proceedings of the IEEE*, vol. 73, no. 4, pp. 523–548, April 1985.

[13] K. Shen and E. J. Delp, "A parallel implementation of an MPEG encoder: Faster than real-time!," *Proceedings of the SPIE Conference on Digital Video Compression: Algorithms and Technologies*, February 5–10, 1995, San Jose, California, pp. 407–418.

[14] M. Tan, J. M. Siegel, and H. J. Siegel, "Parallel implementation of block-based motion vector estimation for video compression on the MasPar MP-1 and PASM," *1995 International Conference on Parallel Processing*, August 14–18, 1995, Boca Raton, Florida, pp. 21–24.

[15] B. Liu and A. Zaccarin, "New fast algorithms for the estimation of block motion vectors," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 3, no. 2, pp. 148–157, April 1993.

[16] A. Puri, H. M. Hang, and D. L. Schilling, "An efficient block-matching algorithm for motion compensated coding," *IEEE International Conference on Acoustics, Speech and Signal Processing*, April 6–9, 1987, Dallas, Texas, pp. 1063–1066.

[17] K. L. Gong, *Parallel MPEG-1 video encoding.* Computer Science Division - Department of EECS, University of California at Berkeley, May 1994. Master's Thesis, Issued as Technical Report 811.

[18] I. Ahmad, K. C. Leung, and H.-M. Hsu, "Multiprocessing ocean circulation: Modeling, implementation, and performance on the Intel Paragon," *Journal of Supercomputing*, vol. 10, no. 4, pp. 349–369, 1997.

[19] Y.-S. Hwang, R. Das, J. H. Saltz, M. Hodoscek, and B. R. Brooks, "Parallelizing molecular dynamics programs for distributed-memory machines," *IEEE Computational Science & Engineering*, vol. 2, no. 2, pp. 18–29, 1995.

[20] J. Wang, P. Liewer, and E. Huang, "Three-dimensional electromagnetic particle-in-cell with monte carlo collision simulations on three MIMD parallel computers," *Journal of Supercomputing*, vol. 10, no. 4, pp. 331–348, 1997.

[21] B. G. Haskell, A. Puri, and A. N. Netravali, *Digital video: an introduction to MPEG-2.* New York: International Thomson Publishing, 1997.

[22] J. M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Transactions on Signal Processing*, vol. 41, pp. 3445–3462, December 1993.

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

[24] V. Bhaskaran and K. Konstantinides, *Image and video compression standards algorithms and architectures*. Massachusetts: Kluwer Academic Publishers, 1995.

[25] C-Cube Microsystems, *C-Cube Microsystems Product Catalog*, Spring, 1994.

[26] R. J. Gove, "The MVP: a highly-integrated video compression chip," *Proceedings of IEEE Data Compression Conference*, March, 28-31 1994, Snowbird, Utah, pp. 215–224.

[27] S. M. Akramullah, I. Ahmad, and M. Liou, "A data-parallel approach for real-time MPEG-2 video encoding," *Journal of Parallel and Distributed Computing*, vol. 30, no. 2, pp. 129–146, November 1995.

[28] G. W. Cook and E. J. Delp, "An investigation of scalable SIMD I/O techniques with application to parallel JPEG compression," *Journal of Parallel and Distributed Computing*, vol. 30, no. 2, pp. 111–128, November 1995.

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

[30] Y. Yu and D. Anastassiou, "Software implementation of MPEG-II video encoding using socket programming in LAN," *Proceedings of SPIE Digital Video Compression on Personal Computers: Algorithms and Technologies*, vol. 2187, February 7–8, 1994, San Jose, California, pp. 229–240.

[31] K. Shen and E. J. Delp, "A spatial-temporal parallel approach for real-time mpeg video compression," *Proceedings of the 25th International Conference on Parallel Processing*, August 13–15, 1996, Bloomingdale, Illinois, pp. II–100–II–107.

[32] L. Sa, V. Silva, L. Cruz, S. Faria, P. Amado, A. Navarro, F. Lopes, and J. Silvestre, "DSP-based hardware for real-time video coding," *Proceedings of the SPIE Conference on Image Processing and Interchange: Implementation and Systems*, vol. 1659, February 12–14 1992, San Jose, California, pp. 99–104.

[33] J. Normile and D. Wright, "Image compression using coarse grain parallel processing," *Proceedings of the 1991 International Conference on Acoustics, Speech, and Signal Processing*, May 14–17 1991, Toronto, Ontario, Canada, pp. 1121–1124.

[34] Y. Okumura, K. Irie, and R. Kishimoto, "Multiprocessor DSP with multistage switching network for video coding," *IEEE Transactions on Communications*, vol. 39, no. 6, pp. 938–946, June 1991.

[35] K. Aono, M. Toyokura, T. Araki, A. Ohtani, H. Kodama, and K. Okamoto, "A video digital signal processor with a vector-pipeline architecture," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 12, pp. 1886–1894, December 1992.

[36] T. Yamazaki, S. Komuro, I. Kumata, and J. Koshiba, "A 1-GOPS CMOS programmable video signal processor," *Digest of Papers - IEEE International Solid-State Circuits Conference*, 1989, pp. 164–165.

[37] J. Goto, K. Ando, T. Inoue, M. Yamashina, H. Yamada, and T. Enomoto, "250-MHz BiCMOS super-high-speed video signal processor (S-VSP) ULSI," *IEEE Journal of Solid-State Circuits*, vol. 26, no. 12, pp. 1876–1884, December 1991.

[38] T. Minami, R. Kasai, H. Yamauchi, Y. Tashiro, J. Takahashi, and S. Date, "A 300-MOPS video signal processor with a parallel architecture," *IEEE Journal of Solid-State Circuits*, vol. 26, no. 12, pp. 1868–1875, December 1991.

[39] T. Inoue, J. Coto, M. Yamashina, K. Suzuki, M. Nomura, Y. Koseki, T. Kimura, T. Atsumo, M. Motomura, B. S. Shih, T. Horiuchi, N. Hamatake, K. Kumagai, T. Enomoto, H. Yamada, and M. Takada, "A 300MHz 16b BiCMOS video signal processor," *Digest of Papers - IEEE International Solid-State Circuits Conference*, 1993, pp. 36–37.

[40] K. Kikuchi, Y. Nukada, Y. Aoki, T. Kanou, Y. Endo, and T. Nishitani, "A single-chip 16-bit realtime video/image signal processor," *Digest of Papers - IEEE International Solid-State Circuits Conference*, 1989, pp. 170–171.

[41] S. Nakagawa, H. Terane, T. Matsumure, H. Segawa, M. Yoshimoto, H. Shinohara, S. Kato, A. Maeda, and Y. Horiba, "A 50ns video signal processor," *Digest of Papers - IEEE International Solid-State Circuits Conference*, 1989, pp. 168–169.

[42] I. Tamitani, H. Harasaki, and T. Nishitani, "A real-time HDTV signal processor: HD-VSP," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 1, no. 1, pp. 35–41, March 1991.

[43] H. Yamauchi, Y. Tashiro, T. Minami, and Y. Suzuki, "Architecture and implementation of a highly parallel single-chip video DSP," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 2, no. 2, pp. 207–220, June 1992.

[44] W. Lee, R. J. Gove, and Y. Kim, "Real-time MPEG video compression using the MVP," *Proceedings of IEEE Data Compression Workshop*, April 1994, Snowbird, Utah.

[45] R. Manepally and D. Sprague, "Intel's i750 (R) video processor—the programmable solution," *Digest of Papers - IEEE Computer Society International Conference -COMPCON '91*, February 25–March 1 1991, San Francisco, California, pp. 324–329.

[46] P. A. Ruetz, P. Tong, D. Luthi, and P. H. Ang, "A video-rate JPEG chip set," *Journal of VLSI Signal Processing*, vol. 5, no. 2/3, pp. 141–150, April 1993.

[47] M. Leonard, "JPEG compression chip cuts system design tasks," *Electronic Design*, pp. 109–113, March 4 1993.

[48] S. Purcell and D. Galbi, "C-Cube MPEG video processor," *Proceedings of the SPIE Conference on Image Processing and Interchange: Implementation and Systems*, vol. 1659, February 12–14 1992, San Jose, California, pp. 24–29.

[49] P. Wayner, "Digital video goes real-time," *Byte*, vol. 19, no. 1, pp. 107–112, January 1994.

[50] S. Purcell, "The C-Cube CL550-JPEG image compression processor," *Digest of Papers - IEEE Computer Society International Conference -COMPCON '91*, February 25–March 1 1991, San Francisco, California, pp. 318–323.

[51] J. Kraus, J. Reimers, and K. Gruger, "A VLSI chip set for DPCM coding of HDTV signals," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 3, no. 4, pp. 302–308, August 1993.

[52] J. Venbrux, P. Yeh, and M. . N. Liu, "A VLSI chip set for high-speed lossless data compression," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 2, no. 4, pp. 381–391, December 1992.

[53] T. Nishitani, I. Tamitani, H. Harasaki, and M. Yano, "A real-time software programmable processor for HDTV and stereo scope signals," *Proceedings of the 1990 International Conference on Application Specific Array Processors*, September 5–7 1990, Princeton, New Jersey, pp. 226–234.

[54] S. K. Rao, M. Hatamian, M. T. Uyttendaele, S. Narayan, J. H. O'Neill, and G. A. Uvieghara, "A real-time P*64/MPEG video encoder chip," *Digest of Papers - IEEE International Solid-State Circuits Conference*, 1993, pp. 32–33.

[55] D. Brinthaupt, L. Letham, V. Maheshwari, J. Othmer, R. Spiwak, B. Edwards, C. Terman, and N. Weste, "A video decoder for H.261 video teleconferencing and MPEG stored interactive video applications," *Digest of Papers - IEEE International Solid-State Circuits Conference*, 1993, pp. 34–35.

[56] S.-M. Lei and M.-T. Sun, "An entropy coding system for digital HDTV applications," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 1, no. 1, pp. 147–155, March 1991.

[57] C.-T. Chiu and K. J. R. Liu, "Real-time parallel and fully pipelined two-dimensional DCT lattice structures with application to HDTV systems," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 2, no. 1, pp. 25–37, March 1992.

[58] K. J. R. Liu and C.-T. Chiu, "Unified parallel lattice structures for time-recursive discrete Cosine/Sine/Hartley transforms," *IEEE Transactions on Signal Processing*, vol. 41, no. 3, pp. 1357–1377, March 1993.

[59] T. Kinoshita and T. Nakahashi, "A 130 Mb/s compact HDTV CODEC based on a motion-adaptive DCT algorithm," *IEEE Journal on Selected Areas in Communications*, vol. 10, no. 1, pp. 122–129, January 1992.

[60] B. D. Alleyne, J. M. Boyce, and I. D. Scherson, "Image block transformations in a partitioned parallel associative processor," *Proceedings of the 1989 International Conference on Parallel Processing*, vol. 3, August 8–12 1989, University Park, Pennsylvania, pp. 48–55.

[61] K. Gaedke, H. Jeschke, and P. Pirsch, "A VLSI based MIMD architecture of a multiprocessor system for real-time video processing applications," *Journal of VLSI Signal Processing*, vol. 5, no. 2/3, pp. 159–169, 1993.

[62] H. Jeschke, K. Gaedke, and P. Pirsch, "Multiprocessor performance for real-time processing of video coding applications," *Journal of VLSI Signal Processing*, vol. 2, no. 2, pp. 221–230, June 1992.

[63] S. P. Yang, A. M. Pleasants, and L. Herron, "The application of enhanced shift encoding and parallel processing to image compression," *Proceedings of the International Conference on Information Engineering*, 1991, pp. 21–25.

[64] M. Tinker, "DVI parallel image compression," *Communications of the ACM*, vol. 32, no. 7, pp. 844–851, July 1989.

[65] 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*, September 1981, Monticello, Illinois, pp. 891–900.

[66] 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*, June 1982, Las Vegas, Nevada, pp. 200–205.

[67] G. W. Cook and E. J. Delp, "The use of high performance computing in JPEG image compression," *presented at the Twenty-Seventh Asilomar Conference on Signals, Systems, and Computers*, November 1–3 1993, Pacific Grove, California.

[68] G. W. Cook and E. J. Delp, "An investigation of JPEG image and video compression using parallel processing," *to be presented at IEEE International Conference on Acoustics, Speech and Signal Processing*, April 19-22 1994, Adelaide, South Australia.

[69] S. M. Akramullah, I. Ahmad, and M. L. Liou, "Performance of software-based MPEG-2 video encoder on parallel and distributed systems," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 7, no. 4, pp. 687–695, August 1997.

[70] Intel Supercomputer Systems Division, Intel Corporation, *Intel Paragon XP/S Technical Summary*, January 1994.

[71] Intel Supercomputer Systems Division, Intel Corporation, *A Touchstone DELTA System Description*, February 26 1991.

[72] K. Jack, *Video Demystified: A Handbook for the Digital Engineer*. California: HighText Publications, Inc., 1993.

[73] I. Pitas, ed., *Parallel Algorithms for Digital Image Processing, Computer Vision and Neural Networks*. Chichester, England: Wiley, 1993.

[74] P. J. Burt and E. H. Adelson, "The Laplacian pyramid as a compact image code," *IEEE Transactions on Communications*, vol. 31, no. 4, pp. 532–540, April 1983.

[75] H. M. Dreizen, "Content-driven progressive transmission of gray level images," *IEEE Transactions on Communications*, vol. COM-35, pp. 289–296, March 1987.

[76] Y. Huang, H. M. Dreizen, and H. P. Galatsanos, "Prioritized DCT for comprression and progressive transmission of images," *IEEE Transactions on Image Processing*, vol. 1, no. 4, pp. 477–487, October 1992.

[77] K. H. Tzou, "Progressive image transmission: a review and comparison," *Optical Engineering*, vol. 26, pp. 581–589, 1987.

[78] K. R. Rao and P. Yip, *Discrete cosine transform: algorithms, advantages, and applications*. Academic Press, 1990.

[79] A. Said and W. A. Pearlman, "A 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.

[80] B. Yazici, M. L. Comer, R. L. Kashyap, and E. J. Delp, "A tree structured Bayesian scalar quantizer for wavelet based image compression," *Proceedings of the 1994 IEEE International Conference on Image Processing*, vol. III, November 13–16 1994, Austin, Texas, pp. 339–342.

[81] C. S. Barreto and G. Mendoncca, "Enhanced zerotree wavelet transform image coding exploiting similarities inside subbands," *Proceedings of the IEEE International Conference on Image Processing*, vol. II, September 16–19 1996, Lausanne, Switzerland, pp. 549–552.

[82] H.-J. Wang and C.-C. J. Kuo, "Novel wavelet coder for color image compression," *Proceedings of SPIE Conference on Multimedia Storage and Archiving Systems II*, November 3–4 1997, Dallas, Texas, pp. 316–327.

[83] D. Taubman and A. Zakhor, "Multirate 3-D subband coding of video," *IEEE Transactions on Image Processing*, vol. 3, no. 5, pp. 572–588, September 1994.

[84] A. N. Netravali and C. B. Rubinsein, "Luminance adaptive coding of chrominance signals," *IEEE Transactions on Communications*, vol. COM-27, no. 4, pp. 703–710, April 1979.

[85] J. O. Limb and C. B. Rubinstein, "Plateau coding of the chrominance component of color picture signals," *IEEE Transactions on Communications*, vol. COM-22, no. 3, pp. 812–820, June 1974.

[86] D. A. Huffman, "A method for the construction of minimum redundancy codes," *Proceedings of the IRE*, vol. 40, pp. 1098–1101, September 1952.

[87] I. Witten, R. Neal, and J. Cleary, "Arithmetic coding for data compression," *Communications of the ACM*, vol. 30, pp. 520–540, 1987.

[88] M. Nelson and J. Gailly, *The data compression book*. M&T Books, 1996.

[89] H. Hotelling, "Analysis of a complex of statistical variables into principal components," *Journal of Educational Psychology*, vol. 24, pp. 417–441 and 498–520, 1933.

[90] C. B. Rubinstein and J. O. Limb, "Statistical dependence between components of a differentially quantized color signal," *IEEE Transactions on Communications Technology*, vol. COM-20, pp. 890–899, October 1972.

[91] P. Pirsch and L. Stenger, "Statistical analysis and coding of color video signals," *Acta Electronica*, vol. 19, no. 4, pp. 277–287, 1976.

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

[93] Q. Wang and M. Ghanbari, "Scalable coding of very high resolution video using the virtual zerotree," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 7, no. 5, pp. 719–727, October 1997.

[94] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies, "Image coding using wavelet transform," *IEEE Transactions on Image Processing*, vol. 1, no. 2, pp. 205–220, April 1992.

[95] M. Ohta and S. Nogaki, "Hybrid picture coding with wavelet transform and overlapped motion-compensated interframe prediction coding," *IEEE Transactions on Signal Processing*, vol. 41, no. 12, pp. 3416–3424, December 1993.

[96] S. A. Martucci, I. Sodagar, T. Chiang, and Y.-Q. Zhang, "A zerotree wavelet video coder," *IEEE Transaction on Circuits and Systems for Video Technology*, vol. 7, no. 1, pp. 109–118, February 1997.

[97] 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*, March 3-6 1996, Belize City, Belize, pp. 162–163.

[98] 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*, September 16–19 1996, Lausanne, Switzerland, pp. Vol II, pp 69–72.

[99] H. Watanabe and S. Singhal, "Windowed motion compensation," *SPIE Conference on Visual Communications and Image Processing*, November 1991, Boston, Massachusetts, pp. 582–589.

[100] K. Shen and E. J. Delp, "A fast algorithm for video parsing using mpeg compressed sequences," *Proceedings of the IEEE International Conference on Image Processin g*, October 23–26 1995, Washington, DC, pp. pp 252–255.

[101] L. H. Jamieson, E. J. Delp, S. E. Hambrusch, A. A. Khokhar, G. W. Cook, F. Hameed, J. N. Patel, and K. Shen, "Parallel scalable libraries and algorithms for computer vision," *Proceedings of the 1994 12th International Conference on Pattern Recognition*, October 9–13 1994, Jerusalem, Israel, pp. 223–228.

[102] K. Shen and E. J. Delp, "Parallel approaches to real-time mpeg video compression," *submitted to Journal of Parallel and Distributed Computing*, 1997.

[103] K. Shen and E. J. Delp, "Wavelet based rate scalable video compression," *submitted to IEEE Transactions on Circuits and Systems for Video Technology*, 1997.

VITA

Ke Shen was born in Shanghai, China, on November 11, 1967. He received a B.S. in Electronics Engineering from Shanghai Jiao-Tong University, China, in 1990. In 1992 he received a M.S. degree in Biomedical Engineering from the University of Texas Southwestern Medical Center at Dallas and the University of Texas at Arlington (jointly). He received a Ph.D. degree in Electrical and Computer Engineering from Purdue University in 1997.

From 1990 to 1991 he worked as a research assistant at UT Southwestern Medical Center at Dallas, and from 1992 to 1997 he worked as a research assistant at Purdue University. He is currently with DiviCom Inc., Milpitas, CA.

His research interests include image/video processing and compression, multimedia systems, and parallel processing. Ke Shen is a member of Eta Kappa Nu.