JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING 30, 129–146 (1995)

A Data-Parallel Approach for Real-Time MPEG-2 Video Encoding[1]

SHAHRIAR M. AKRAMULLAH,* ISHFAQ AHMAD,† AND MING L. LIOU*

*Department of Electrical and Electronic Engineering and †Department of Computer Science, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong

In this paper, we present a fine-grained parallel implementation of the MPEG-2 video encoder on the Intel Paragon XP/S parallel computer. We use a data-parallel approach and exploit parallelism within each frame, unlike some of the previous approaches that employ multiple processing of several disjoint video sequences. This makes our encoder suitable for real-time applications where the complete video sequence may not be present on the disk and may become available on a frame-by-frame basis with time. The *Express* parallel programming environment is employed as the underlying message-passing system making our encoder portable across a wide range of parallel and distributed architectures. The encoder also provides control over various parameters such as the number of processors in each dimension, the size of the motion search window, buffer management, and bitrate. Moreover, it has the flexibility to allow the inclusion of fast and new algorithms for different stages of the codec into the program, replacing current algorithms. Comparisons of execution times, speedups, and frame encoding rates using different numbers of processors are provided. An analysis of frame data distribution among multiple processors is also presented. In addition, our study reveals the degrees of parallelism and bottlenecks in the various computational modules of the MPEG-2 algorithm. We have used two motion estimation techniques and five different video sequences for our experiments. Using maximum parallelism by dividing one block per processor, an encoding rate higher than 30 frames/s has been achieved. © 1995 Academic Press, Inc.

1. INTRODUCTION

A codec consists of an encoder and a decoder. For ease of storage and transport of large amount of video data, major functions of a video codec are compression and decompression which are performed by the encoder and the decoder, respectively. These operations require a great deal of signal processing of the order of billion operations per second. Although a video sequence has to be displayed in real-time, depending on applications, video encoding can be done either in real-time or in non-real-time. Since video encoding is much more complex and time-consuming compared to decoding, it is always advantageous to speedup the computation. Usually, special hardware (VLSI) is needed to implement encoder for real-time applications. On the other hand, software implementation of encoder is extremely difficult for real-time applications. However, for prototyping and pre-recorded compression (non-real-time), it is very desirable and effective. This paper explores this possibility of software implementation of a video codec, in particular, MPEG-2, by using parallel computers with a view to achieve a high speedup in computation. MPEG-2 [13] is the most recent video coding standard established by the Motion Pictures Expert Group of the International Standards Organization (ISO). Being very comprehensive it is still evolving and is therefore chosen for our software implementation.

MPEG-2 embodies different modules some of which are very computation intensive. Since MPEG-2 is designed as a *generic* standard to support a wide variety of applications, several bit-rates of 2 Mbps and up, and various qualities and services, it often needs to deal with processing of large amount of data, for example, 10.37 million pels/s according to CCIR 601 specification. Since the encoding time is proportional to the total number of pixels to process, the encoder requires extensive computation to fully support applications such as HDTV transmission with embedded standard TV, video-on-demand (VOD), cable TV distribution, video communications on asynchronous transfer mode (ATM) networks, digital terrestrial television broadcast, home television theater, multimedia mailing, remote video surveillance, satellite news gathering and networked database services, etc. MPEG-2 requires that it should also include the functionalities such as normal play, fast forward/reverse, random access, and normal reverse [3]. Since it is very slow when performed on a conventional serial computer, a parallel implementation of the encoder is an obvious and most promising software solution.

There have been some previous approaches—mostly using special hardware—to parallelize codec operations of video sequences. For example, the CD-I full-motion video encoding (nonstandard) has been implemented on a parallel computer [25]. It employs an approach where parallelization is done by dividing the stages of the video codec into tasks and assigning one task to a group of processors. This has the disadvantage that many frames must be read

[1] This research was supported in part by the Hongkong Telecom Institute of Information Technology (HKTIIT) and in part by the Student Financial Assistance Agency, Hong Kong, under the Commonwealth Scholarship and Fellowship Plan, 1993–1995.

0743-7315/95 $12.00

before all processors have some tasks to execute. Furthermore, this implementation requires a special-purpose hardware.

An MPEG-1 encoder has been implemented in [27] with a special-purpose parallel hardware using DSP processors. This has the drawback that the parallel programs are nonportable and must match the ability of the special hardware which has limited capability in signal processing. Another hardware implementation of MPEG-2 using image compression DSP with good performance has been described in [1].

A parallel implementation of the H.261 video coding algorithm using a single-instruction multiple-data (SIMD) parallel machine has been reported in [18]. It suggests that parallelization can be done either on individual pixels, on blocks or on other groups of the frame data, while this implementation chose to distribute data-blocks among processors. However, due to the nature of SIMD paradigm, only certain parts of H.261 were parallelized. Consequently the implementation of rate control became the bottleneck in processing and could only achieve a frame rate of about 5 frames/s. Distributed load balancing schemes for parallel video encoding system have been described in [11], which involve a hybrid video encoding algorithm recommended by CCITT SG XV for $p \times 64$ kbits/s.

Parallel MPEG-1 video encoding with a performance of about 4 frames/s using 9 HP 9000/720 machines connected via ethernet has been documented in [10]. It has been later modified as described in [24] to run on Intel Touchstone Delta and Intel Paragon. Although this implementation has accomplished faster than real-time performance, but it has several drawbacks. First, it divides the video sequence into different sections and assignes those sections to different processors. Each processor runs the same sequential encoding program, but compresses different parts of the video (in particular, different Group of Pictures) in parallel. Therefore, the complete video sequence should be available before encoding begins. Second, the compressed data from each processor has to be concatenated off-line. Third, since it was found inefficient to open a large number of small files for input and output, this implementation has to group sections of consecutive frames into a single file. To get the most efficient performance using this approach, a processor should open and read from a single file having all the frames assigned to it, including the necessary reference frames. This means one can use only a limited number of processors to encode a video sequence of a given length, which restricts the scalability of the problem. Finally, it uses some special I/O operation capability offered by Intel Touchstone Delta or Intel Paragon in order to improve performance, and therefore is not portable to other hardware platforms, for instance, a network of workstations. A slice-based software implementation of MPEG-2 video encoding is described in [29]. This implementation uses socket programming for a local area network (LAN) of workstation for its parallelism. For our implementation of MPEG-2, we have chosen the data-parallel approach similar to [18] on a multiple-instruction multi-data (MIMD) machine. Our implementation does not employ any special-purpose hardware or programming primitives, rather it is completely portable, flexible, and scalable. The implementation is administered on the Intel Paragon, as well as Intel iPSC/860 and various types of networks of workstations, although only the results obtained on Intel Paragon are reported here.

The rest of the paper is arranged as follows. Section 2 gives a description of MPEG-2, mentioning its differences or improvements with respect to its previous counterparts. Section 3 introduces the Intel Paragon XP/S. Section 4 describes the methodology adopted for the current implementation. It also discusses the data distribution and communication strategies and provides an expression of maximum number of processors that can be used. Section 5 provides experimental results. The last section gives some future research directions and conclusions.

2. AN OVERVIEW OF MPEG-2

The ISO/IEC 13818-2 standard, commonly known as MPEG-2 video standard specification, provides a standard syntax which offers an optimum between cost—for example, VLSI area, memory size and bandwidth—and quality such as compression efficiency. So it is rapidly emerging as the preferred standard for full motion video compression.

MPEG-2 may be considered as the enhanced version of ISO/IEC 11172-2 video coding standard, commonly known as MPEG-1 [15]. MPEG-1 was established for digital storage applications with VCR quality at 1.5 Mbps. MPEG-2 supports a variety of applications, generally with higher picture quality at a bit-rate greater than or equal to 2 Mbps. It is also compatible with MPEG-1. A special case of MPEG-2 is H.262 which is intended for high quality visual communication.

2.1. Structure of MPEG-2

The bitstream syntax of the MPEG-2 standard [13] is divided into subsets known as profiles, which specify constraints on the syntax. The profiles are again divided into sets of constraints imposed on parameters in the bitstream, which are known as levels. There are five profiles defined by MPEG-2: simple, main, SNR scalable, spatially scalable, and high; some of these involve upto four levels: low, main, high 1440, and high. In each level there may be three formats of color spaces, representing luminance and two chrominance distributions, namely, 4:2:0, 4:2:2, and 4:4:4. Also, in addition to the base layer in a level, there may be one or more enhancement layer. The profiles and levels of MPEG-2 limit the syntax and operation of its various components and the parameters, for example, sample rates, frame dimensions, coded bitrates, etc. (see Table I), respectively. Thus they optimize computational complexity, buffer size, and memory bandwidth while still addressing the widest possible range of applications.

TABLE I
Picture Quality in Terms of PSNR

Parameters	Profiles				
	Simple	Main	SNR Scalable	Spatially Scalable	High
Horizontal size (≤ pels)	720	352 (l) 720 (m) 1440 (h) 1920 (h)	352 (l) 720 (m)	1440 (e) 720 (b)	720 (m,e) 352 (m,b) 1440 (h,e) 720 (h,b) 1920 (h,e) 960 (h,b)
Vertical size (≤ pels)	576	288 (l) 576 (m) 1152 (h) 1152 (h)	288 (l) 576 (m)	1152 (e) 576 (b)	576 (m,e) 288 (m,b) 1152 (h,e) 576 (h,b) 1152 (h,e) 576 (h,b)
Frame rate (≤ frames/sec)	30	30 (l) 30 (m) 60 (h) 60 (h)	30 (l) 30 (m)	60 (e) 30 (b)	30 (m) 60 (h,e) 30 (h,b) 60 (h,e) 30 (h,b)
Total number of pixels (≤ pel/sec)	10, 368, 000	3, 041, 280 (l) 10, 368, 000 (m) 47, 001, 600 (h) 62, 668, 800 (h)	3, 041, 280 (l) 10, 368, 000 (m)	47, 001, 600 (e) 10, 368, 000 (b)	11, 059, 200 (m,2,e) 3, 041, 280 (m,2,b) 14, 745, 600 (m,0,e) 3, 041, 280 (m,0,b) 47, 001, 600 (h,2,e) 11, 059, 200 (h,2,b) 62, 668, 800 (h,0,e) 14, 745, 600 (h,0,b) 62, 668, 800 (h,2,e) 14, 745, 600 (h,2,b) 83, 558, 400 (h,0,e) 19, 660, 800 (h,0,b)
Bit rate (≤ bits/sec)	15, 000, 000	4, 000, 000 (l) 15, 000, 000 (m) 60, 000, 000 (h) 80, 000, 000 (h)	4, 000, 000 (l,a) 3, 000, 000 (l,b) 15,000,000 (m,a) 10,000,000 (m,b)	60, 000, 000 (a) 40, 000, 000 (e) 15, 000, 000 (b)	20, 000, 000 (m,a) 15, 000, 000 (m,m) 4, 000, 000 (m,b) 80, 000, 000 (h,a) 60, 000, 000 (h,m) 20, 000, 000 (h,b) 100, 000, 000 (h,a) 80, 000, 000 (h,m) 25, 000, 000 (h,b)
Ratio of bit rate to frame rate (≤ kbits)	626	167 (l) 626 (m) 2, 503 (h) 3, 337 (h)	167 (l,a) 126 (l,b) 626 (m,a) 418 (m,b)	2, 503 (a) 1, 669 (m) 626 (b)	835 (m,a) 626 (m,m) 167 (m,b) 3, 337 (h,a) 2, 503 (h,m) 835 (h,b) 4, 171 (h,a) 3, 337 (h,m) 1, 043 (h,b)
VBV Buffer Size (≤ bits)	1, 835, 008	489, 472 (l) 1, 835, 008 (m) 7, 340, 032 (h) 9, 787, 392 (h)	489, 472 (l,a) 367, 616 (l,b) 1, 835, 008 (m, a) 1, 223, 680 (m, b)	7, 340, 032 (a) 4, 893, 696 (m) 1, 835, 008 (b)	2, 447, 360 (m,a) 1, 835, 008 (m,m) 489, 472 (m,b) 9, 787, 392 (h,a) 7, 340, 032 (h,m) 2, 447, 360 (h,b) 12, 233, 728 (h,a) 9, 787, 392 (h,m) 3, 036, 160 (h,b)

[a] here l = low level, *m* = main level, *h* = high 1440 level, h = high level, e = enhancement layer, b = base layer, a = all layer, m = middle layer, 2 = 4:2:2 format, 0 = 4:2:0 format.

The bitstream syntax can also be divided into two major categories: the first is the nonscalable syntax (a superset of MPEG-1) featuring the extra compression tools for interlaced video signals along with some other important inclusions such as variable bit-rate, alternate scanning, concealment motion vectors, etc.; and the second is the scalable-syntax, having property of enabling the reconstruction of useful video from pieces of a total bitstream by structuring the bitstream in a base layer and some enhancement layers, where the base layer can use the non-scalable syntax. The purpose of layered structure is to facilitate the decoding process and to prevent ambiguity while it sup-

ports the claims of *genericity*, *flexibility*, and *efficiency* [17]. There is an abstract model of decoding used to verify that an MPEG bitstream is decodable within reasonable buffering and delay requirement, which is known as *video buffering verifier* (VBV).

The coded representation defined in the non-scalable syntax can achieve a high compression ratio while preserving good picture quality. Since the exact pixel values are not preserved during coding, the algorithm is not lossless. The choice of the techniques is based on the need to balance a high picture quality and compression ratio with the requirement to make random access to the coded bitstream. Obtaining good picture quality at the bitrates of interest demands very high compression, which is not achievable with intraframe coding alone. The random access requirement, however, is best satisfied with pure intraframe coding. This necessitates a delicate balance between intra- and interframe coding and between recursive and nonrecursive temporal redundancy reduction, leading to the definition of Intracoded (I), Predictive coded (P), and Bidirectionally predictive coded (B) pictures. This idea is illustrated in Fig. 1.

Here, I-pictures provide good random access with moderate compression and are used as reference pictures for future prediction. P-pictures are coded more efficiently from a previous I- or P-picture and are generally used as reference pictures for further prediction. B-pictures provide the highest degree of compression but require both past and future reference pictures for motion compensated prediction. The display order of these frames (*I*, *B*, *B*, *P*, *B*, ...), for the example shown in Fig. 1 would be different from the transmission order (*I*, *P*, *B*, *B*, *P*, ...).

The algorithm first selects an appropriate spatial resolution for the signal. It then uses block-based motion-compensated prediction for the temporal redundancy reduction, which falls into the temporal DPCM (differential pulse code modulation) category. Motion compensation is used both for causal prediction of the current picture from a previous reference picture, and for noncausal, interpolative prediction from past and future reference pictures. Next in the algorithm is the stage of motion estimation, which covers a set of techniques used to extract the motion information from a video sequence. Motion vectors are defined for each 16-pixel by 16-line region of the picture. In MPEG-2, motion estimation is done by using block-matching technique. Figure 1 shows an example of block matching. Here, for block B in the current frame, the best match is with block A in the previous frame and with block C in the future frame.

In order to achieve spatial redundancy reduction, the

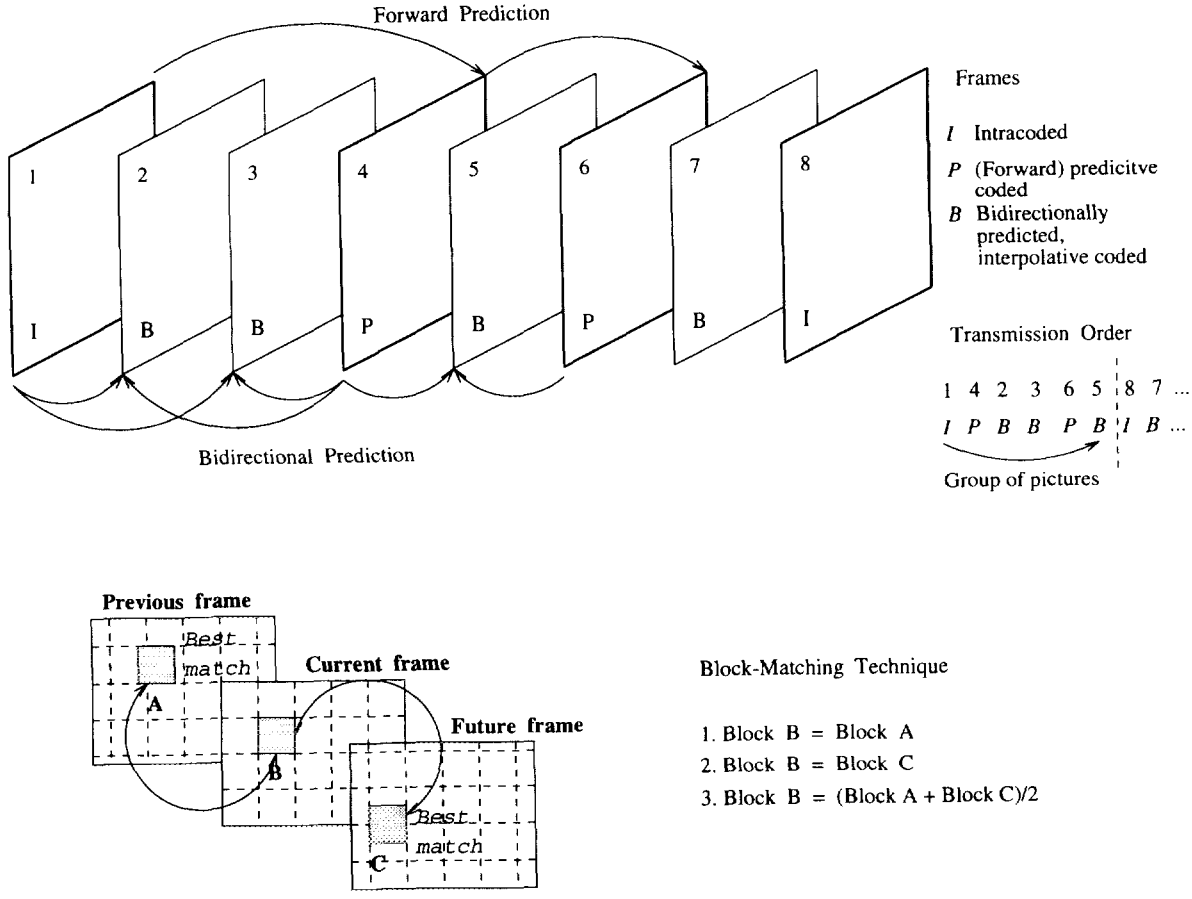

FIG. 1. Typical arrangement of group of pictures in MPEG-2.

difference signal, i.e., the prediction error, is further compressed using the block transform coding technique which employs the two-dimensional orthonormal 8 × 8 DCT (discrete cosine transform) to remove spatial correlation. The resulting 63 AC transform coefficients are mapped in an alternate scanning pattern (or zig-zag scanning pattern when providing compatibility to MPEG-1) before it is quantized in an irreversible process that discards the less important information. Luminance and chrominance components of the frame data share the same quantization tables. In MPEG-2, adaptive quantization is used at the macroblock (16 × 16 pel area) layer, which pemits smooth bit-rate control as well as perceptually uniform quantization throughout the picture and image sequence. Finally, the motion vectors are combined with the residual DCT information, and transmitted using variable length codes. The variable length coding tables are non-downloadable and are therefore optimized for a limited range of compression ratios appropriate for the target applications. A block diagram of the complete encoder is given in Fig. 2.

In the scalable syntax, scalability tools support applications beyond that supported by single layer video, by breaking the bitstream into different layers which prioritize the video data. For example, the high priority channel (bitstream) can be coded with a combination of extra error correction information and decreased bit-error (i.e., higher signal strength) than the lower priority channel. Another purpose of scalability is the division of complexity. Complexity division could be simply obtained by simulcasting technique which is based on transmission/storage of multiple independently coded reproductions of video. Nevertheless, a more efficient alternative is scalable video coding, in which the bandwidth allocated to a given reproduction of video can be partially reutilized in coding of the next reproduction of the video. The basic scalability tools offered by MPEG-2 are:

- *Spatial Scalability.* This is intended for use in video systems for which a minimum of two layers of spatial resolution are necessary. This spatial domain method codes a base layer at lower sampling dimensions (i.e., "resolution") than the upper layers. The upsampled reconstructed lower (base) layers are then used as prediction for the higher layers.
- *Data Partitioning.* This in intended for use when two channels are available for transmission and/or storage of a video bitstream. Data partitioning is a frequency domain method that breaks the block of 64 quantized transform coefficients into two bitstreams. The first, higher priority bitstream contains the more critical lower frequency coef-

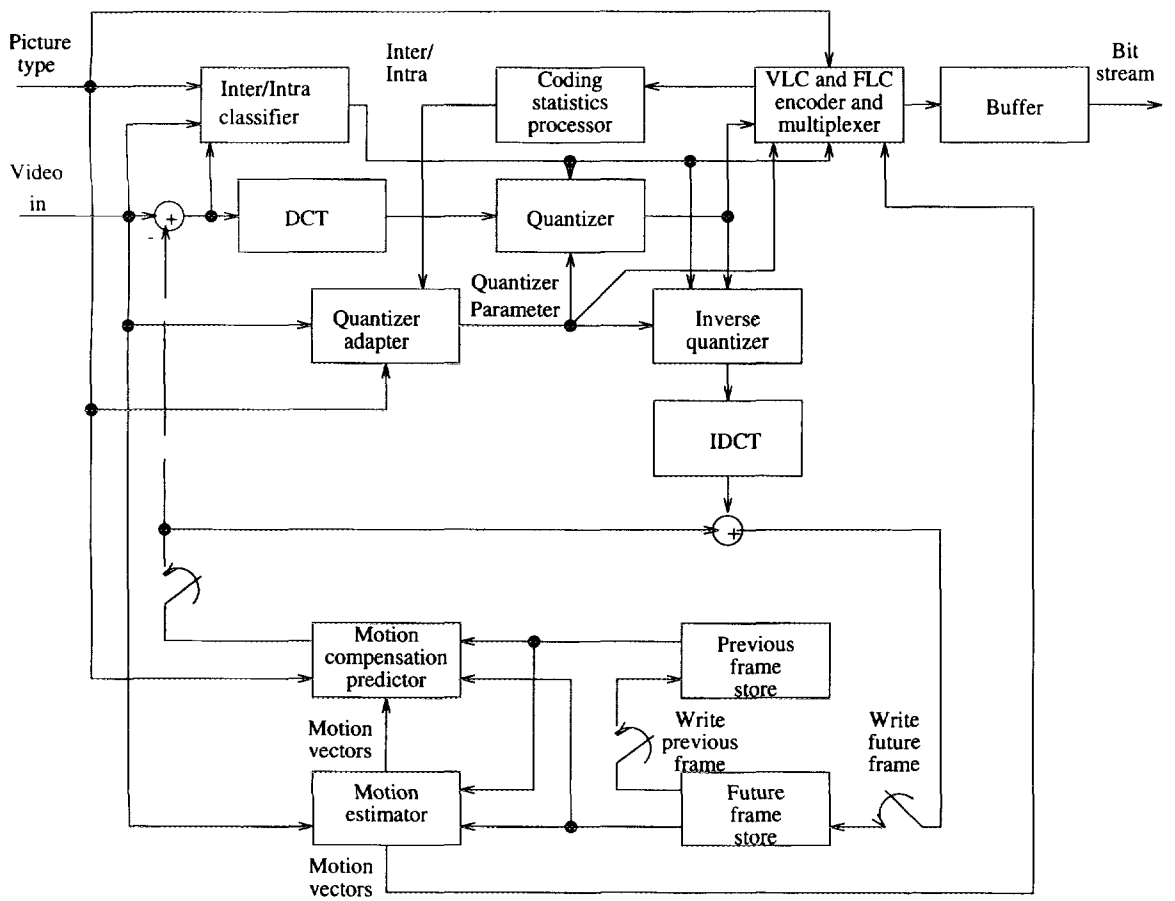

FIG. 2. The MPEG-2 computational modules.

ficients and side informations (such as DC values and/or motion vectors). The second, lower priority bitstream carries higher frequency AC data.

• *SNR Scalability.* SNR scalability is a spatial domain method where channels are coded at identical sample rates, but with different picture quality (through quantization step sizes). The higher priority bitstream contains base-layer data that can be added to a lower priority refinement layer to regenerate a higher quality reproduction of the input video.

• *Temporal Scalability.* A temporal domain method is useful in applications for which migration to higher temporal resolution systems may be necessary. The first, higher priority bitstream codes video at a lower frame-rate, and the intermediate frames can be coded in a second bitstream using the first bitstream reconstruction as prediction. It provides transmission error resilience as the more important base layer data can be transmitted over a channel with better error performance.

2.2. Range of Parameters

Table 1 shows the limits of different parameters permitted by MPEG-2. To deal with various applications of different nature, MPEG-2 specifies these constraints, by decomposing the bit-stream syntax into subsets called profiles and levels, as explained in the previous section. The parameters to which the constraints are entangled are mainly: horizontal size of a frame; vertical size of a frame; the frame rate; the product of the horizontal size, the vertical size, and the frame rate; the coded data rate (bit-rate); and the VBV (video buffering verifier) buffer size.

3. AN OVERVIEW OF INTEL PARAGON XP/S

The Paragon XP/S [12] from Intel Corporation was first delivered in September 1992. Similar to its predecessors the Touchstone Delta (prototypical) and the iPSC/860, the Paragon is also a multicomputer. Its nodes are based on Intel's i860 XP processor and it primarily supports message passing as the programming model using a new OSF/1-based operating system.

From an architectural point of view, the Paragon is a distributed-memory MIMD machine. Its processing nodes are arranged in a two-dimensional rectangular grid. The system consists of three types of nodes: compute nodes, which are used for the execution of parallel programs; service nodes, which offer capabilities of a UNIX system, including compilers and program development tools; and I/O nodes, which are interface to mass storage and LANs. All nodes are uniformly integrated in the interconnection network. The login sessions are distributed in a user-transparent manner to one of the nodes in service partition from where jobs can be initiated interactively or submitted via network queuing system (NQS). The I/O nodes provide ethernet and high performance parallel interface (HiPPI) connections to the Paragon; a fibre distributed data interface (FDDI) network can be attached to a HiPPI node by using a router.

Paragon mesh routing chips (MRCs), connected by high-speed channels, are the basis of the communication network, where nodes may be attached. These are two independent channels—one for each direction—between two neighboring nodes. The channels are 16 bits wide and have a theoretical bandwidth of 175 Mbps. The MRCs can route messages autonomously and are independent of the attached nodes. Communication is based on *wormhole* routing with a deterministic routing algorithm [4]. Messages are sent first in the horizontal and then in the vertical direction. Although the pipelined nature of the wormhole routing allows the bandwidth to be nearly independent of the distance between the communicating nodes, due to software overhead, it is limited to 90 Mbps for system release 1.2.

All the three types of nodes are implemented by the same general purpose (GP) node hardware. A 32-bit address bus and a 64-bit, 50 MHz (i.e., 400 Mbps) data bus connects all the components of the GP node's compute and network interface parts. The i860 XP is a fast compute-oriented reduced instruction set computer (RISC) processor with a clock speed of 50 MHz and a theoretical peak performance of 75 Mflops (64-bit arithmetic: 50 Mflops add, 50 Mflops multiply) and 100 Mflops (32-bit arithmetic: 50 Mflops add, 25 Mflops multiply). When used in parallel, *add* and *multiply* do not operate at full speed. The integer peak performance is 42 Mips (32-bit) including load/store. The interface of the GP node to the interconnection network performs such that message-passing is separated from computing. The actual transmission of data between the memory and the network is accomplished by the network interface controller (NIC).

The Paragon's primary mass storage system is a disk array like redundant array of inexpensive disks (RAIDs) which can accommodate 4.2 Gbyte of data. The node memory is organized in two banks and constructed on the basis of 4 Mbit, 60 ns DRAM chips. The peak speed of data transfer between the memory and the processor caches is 64 bits per cycle, i.e., 400 Mbps.

The Paragon's operating system is called Paragon OSF/1, which provides an OSF/1-compatible application interface. On the compute nodes, the operating system and related buffers occupy more than 6 Mbyte. The OSF/1 servers and libraries jointly offer the view of a single UNIX-like system to the user. The NX message passing interface used in Paragon is a super-set of the iPSC/860's NX/2. Paragon OSF/1 supports virtual memory on all nodes. The file system of Paragon OSF/1, called the parallel file system (PFS), is upward compatible with the concurrent file system (CFS) of the iPSC/860. It is based on a single-processor file system, namely, the Berkeley 4.3 virtual file system (VFS), along with a distributed file system. PFS provides parallel access to a single file and higher data transfer rates by striping files across multiple I/O nodes and their attached RAID systems. It allows processor mesh to be

divided into sets of nodes, called partitions, thus providing a way to restrict part of the mesh for particular users or types of jobs and a mode of specifying different scheduling characteristics on different segments of the machines.

The Paragon system primarily supports the message-passing programming model with the NX library. While support is provided for both synchronous and asynchronous messages as well as interrupt-producing messages, global operations such as *global sum* are also available. Several tools, including some runtime profiling tools, are available for performance analysis. Portable message-passing libraries such as EXPRESS, PVM, PARMACS, and TCGMSG can be used on the Paragon.

For our experiments, we have used a 140-node Paragon at the Hong Kong University of Science and Technology, Hong Kong, and a 512-node Paragon at the California Institute of Technology.

4. METHODOLOGY FOR PARALLEL IMPLEMENTATION

The parallel implementation of MPEG-2 has been carried out using a data-parallel or SPMD programming paradigm. The SPMD paradigm written in the C programming language under Express [21] allows our software to be portable across a wide range of architecture ranging from high-performance scalable systems to high-speed networks of workstation. In order to make our parallel MPEG-2 implementation scalable, we assume that our target processor topology is a two-dimensional grid. This has been achieved using Express's Cubix programming model which, in addition to providing the facility of overlapped data reading, can set up a virtual processor grid regardless of the hardware topology and then automatically map the data onto this array of processors. This allows us to control the granularity of the problem by enabling it to run on a few fast workstation in a coarse-grained fashion as well as large-scale fine-grained massively parallel processing systems.

Another reason for choosing this programming environment is that it provides all the communication primitives and several library routines, which are very easily pluggable to a normal C code. In addition, not only does it contribute to a minimal programming effort, but also the program does not need to use machine-specific constructs for message-passing. Furthermore, it does not rely on a specific network topology or number of processors. These countenances enhance the portability of the program to other MIMD computers.

The following format of the input frame data (as per MPEG-2) is considered: There are three input streams, the luminance (Y) pixel stream, and two chrominance (Cb and Cr) pixel streams. Each input stream represents the line-by-line scan of the frame. The frame data is then distributed among the processors, each processor having some 8×8 blocks of data, depending upon the number of processors available. Unlike [29], which uses a *slice* as the basic unit for parallel processing, our approach rather resembles [18]. Motion estimation is performed on independent macroblocks (16×16) while other operations used 8×8 blocks as the basic unit of parallel processing.

4.1. Data Distribution

Since the overhead due to indispensable communication can be the major limiting factor, care should be taken while partitioning the data among the processors such that minimal interprocessor communication is employed. The data-distribution to the processors for the data-parallel approach is very simple. The whole frame is distributed as evenly as possible to each processor. It is possible to partition the data by just apportioning the requisite part of the frame data (one or more 16×16 macroblock) to the corresponding processors as the processors are mapped onto the two dimensional grid. This method is shown in Fig. 3a. But in that case, it necessitates inevitable communication between processors as the search window during motion estimation would move to the boundary (see Fig. 4). On the contrary, since each processor has enough memory to store the entire search window, it is possible to eliminate use of overwhelming amount of communication. In this case, the frame data is distributed among the processors allowing overlap and using the data partitioning method as depicted in Fig. 3b. Here, each processor is allocated some redundant data, which is necessary to form the complete search area. Let us consider P and Q to be the height and the width of the frame, respectively, and let p be the total number of processors to be used, with p_h to be the number of processors in the horizontal dimension and p_v to be the number of processors in the vertical dimension. Thus, $p = p_h \times p_v$. If the search window size is the size of the macroblocks in a particular processor $\pm W$ in both dimensions, with overlapped (redundant) data distribution, given p_h and p_v, one can determine the size of the local frame in each processor, which is given by

$$X_{\text{local}} = \left\lceil \frac{Q}{p_h} + 2W \right\rceil \times \left\lceil \frac{P}{p_v} + 2W \right\rceil. \tag{1}$$

It is easy to see that if we want to avoid interprocessor communication when computing motion estimation, some additional memory is required for every processor to accommodate the redundant data necessary to form the search window. This is a realization of the popularly known communication-memory trade-off. In our implementation, the number of processors to be used is an input parameter. Therefore, it can be ported to environments with a few powerful processors to those with a large number of relatively slow processors as well as to hardware platforms with limited memory or slow communication.

4.2. An Upper Bound for Number of Processors

The maximum number of processors that may be used can be determined as follows. Let the macroblock size be

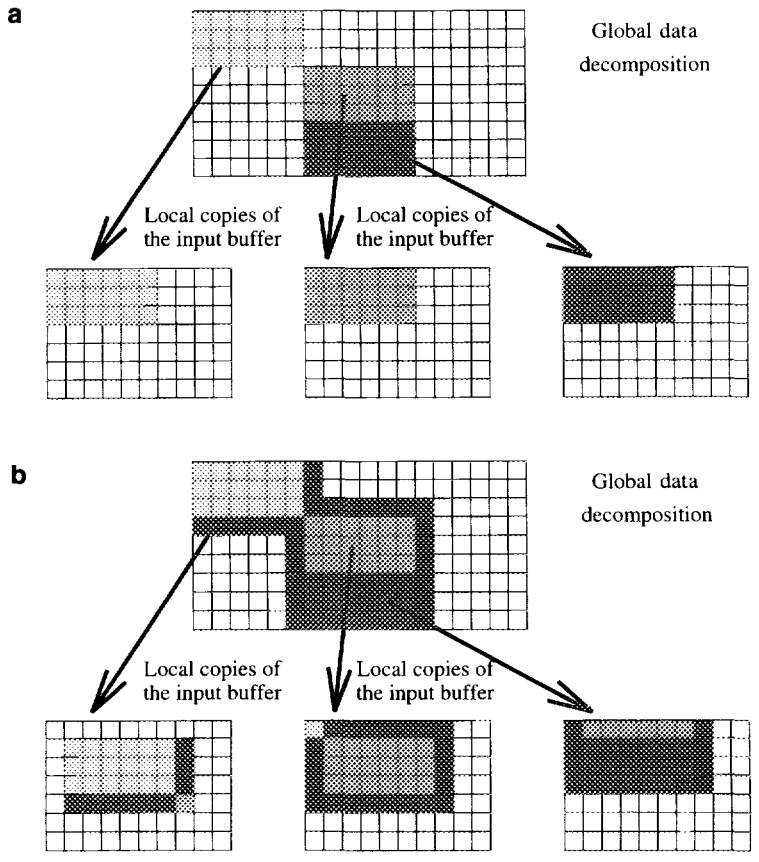

FIG. 3. (a) Data distribution without overlapping. (b) Data distribution with overlapping.

$w \times h$ (typically 16×16) for which motion vectors are to be determined. The frame data may be divided among processors if we can make the search window available to the corresponding processor. This can be accomplished by distributing one or more 16×16 block to each processor, and then either sending the required boundary data to corresponding processors to form their local frame (see Fig. 4), that is search window, or equivalently, storing the redundant data at the local memory of each processor, which is necessary to form the search window to be used for determination of motion vectors for the next frame. Both types of distribution of data are discussed in the previous section. Thus, one can have

$$p_{\text{h,max}} = \left\lfloor \frac{Q}{w} \right\rfloor \quad \text{and} \quad p_{\text{v,max}} = \left\lfloor \frac{P}{h} \right\rfloor. \tag{2}$$

Hence, for $w = h = 16$, maximum number of processors is

$$p_{\max} = \left\lfloor \frac{Q \times P}{16 \times 16} \right\rfloor \Rightarrow p \leq \left\lfloor \frac{Q \times P}{256} \right\rfloor. \tag{3}$$

For example, if $Q = 360$, $P = 240$, then $p_{\text{h,max}} = 22$, $p_{\text{v,max}} = 15$ and consequently $p \leq 330$.

4.3. Implementation Feature

Our implementation of the MPEG-2 encoder generates constant bit-rate bit-streams and supports *progressive* as well as *interlaced* video. It is also able to generate MPEG-1 bit-streams. It supports three input formats: separate YUV, combined YUV, and PPM (Portable PixMap format [23]). It outputs the encoded sequence as well as relevant statistics and verifies legality of the user-given parameters within profile and level. The current implementation does not support variable bit rate encoding, scalable extensions, integer pel motion vectors for MPEG-1 (always produces half-pel motion vectors, which give better quality), 3:2 pull down, low-delay, concealment motion vectors, editing of encoded video, and scene change rate control. Our parallel implementation is based on a sequential implementation [19] of MPEG-2 video coding algorithm.

4.3.1. DCT and IDCT

Since DCT approaches the statistically optimal Karhunen–Loeve transform (KLT) for highly correlated signals such as pixels in a picture frame, it is widely used in motion video or still image codec applications [6]. For this reason DCT has been adopted in MPEG-2 as well as in other standards. Compression is achieved owing to the following

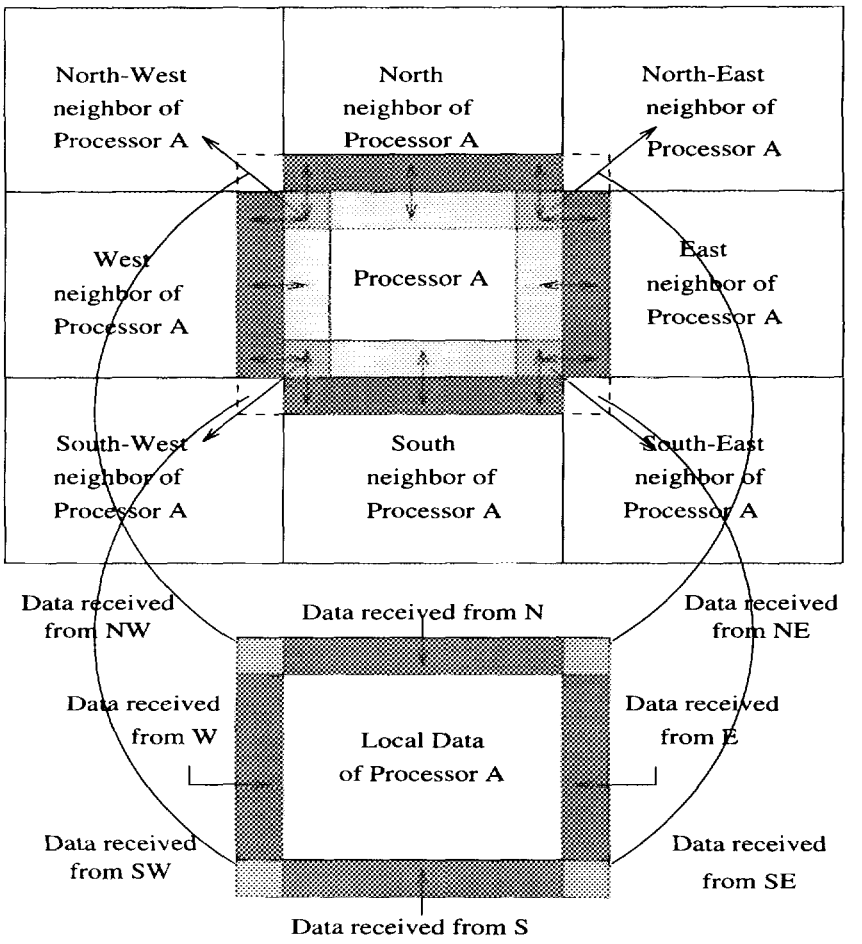

FIG. 4. The flow of data from a processor to neighboring processors.

fact. The data, transformed using DCT, is scanned in an alternate zig-zag scanning pattern and then quantized. The high frequency components, which are insignificant to human visual perception, are discarded in the process. This provides a large run of zeros, which can be very efficiently used in run-length coding to yield data-compression.

The $N \times N$ two-dimensional DCT is defined as

$$F(u,v) = \frac{2}{N} C(u)C(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) \cos\frac{(2x+1)u\pi}{2N} \cos\frac{(2y+1)v\pi}{2N} \tag{4}$$

with $u, v, x, y = 0, 1, 2, \ldots, N-1$, where x, y are spatial coordinate in the pixel domain, u, v are coordinates in the transform domain and

$$C(u), C(v) = \begin{cases} \frac{1}{\sqrt{2}}, & \text{for } u, v = 0, \\ 1, & \text{otherwise.} \end{cases}$$

The inverse-DCT (IDCT) is defined as

$$f(x,y) = \frac{2}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} C(u)C(v)F(u,v) \cos\frac{(2x+1)u\pi}{2N} \cos\frac{(2y+1)v\pi}{2N}. \tag{5}$$

In our implementation with full-search motion estimation, for DCT, standard *row–column approach* is used, while for IDCT, *Wang's fast algorithm* [28] is used. On the other hand, for the implementation that used the logarithmic search for motion estimation, we used Wang's algorithm with double precision for both DCT and IDCT. The DCT and IDCT are performed on the 8×8 pixel blocks. The same serial program is executed on each processor to compute DCT or IDCT for as many blocks belonging to its local share of the frame data. So there is no interprocessor data movement.

4.3.2. Motion Estimation

When a moving object exists in a video sequence, the problem of motion estimation, that is to estimate displace-

ments of moving object from frame to frame, is encountered. In motion-compensated coding (MCC), which is adopted to MPEG, the problem of motion estimation needs special attention. Basically, there are two different approaches [20] for motion estimation: *pel recursive algorithm* (PRA) which estimates recursively the displacement of each pel from its neighboring pels, and *block matching algorithm* (BMA) which finds the best match for a block (according to MPEG, 16 × 16) belonging to the current frame, within a search area in the previous frame. Although the performance (accuracy of motion estimation) offered by the former method is higher, the latter is still widely used owing to relative ease of implementation and relative low computational cost, (which is still so high that it leads to the motivation of parallel implementation), and is adopted in MPEG.

In general, block-matching algorithm proceeds as follows: the motion vector is obtained by minimizing a cost function by means of measurement of the mismatch between a block and each predictor candidate. Let M be a macroblock (16 × 16) in the current picture, X_c, v is the displacement with respect to the corresponding macroblock in the reference picture X_r, then the optimal displacement (motion vector) is obtained by the formula

$$\bar{v}^* = \min \sum_{\bar{x} \in M} D[X_c(\bar{x}) - X_r(\bar{x} + \bar{v})], \quad \bar{v} \in V, \qquad (6)$$

where X_c is the pixel value in the current frame, X_r is for the previous frame, and V is the search range of the possible motion vectors. The value of V and the selection of the cost function D are left entirely to the implementation [17].

In our case, mean absolute difference (MAD) is chosen as the matching criterion (D) on the reference frame, while the search range V remains as an input parameter. The minimum mean absolute difference ($MMAD$) is given by,

$$MMAD = \min_{i,j} \left(\frac{1}{mn} \sum_k \sum_l |X_c(k, l) - X_r(k + i, l + j)| \right), \qquad (7)$$

where (k, l) is the location of the m-by-n block in the current frame, and the amount of shift is i pels and j lines in the direction of scanning within the search area. The motion vector with minimum MAD is considered as the best motion vector.

We have employed both exhaustive and fast-search patterns for motion estimation in our implementation. The two-dimensional logarithmic search [16] is used as the fast motion search technique. Exhaustive search examines all motion vectors in the search window while there is a significant reduction of number of motion vectors to search when employing logarithmic search. The logarithmic search is based on the assumption that the matching criterion (D) increases monotonically as the search moves away from the direction of minimum error. This direction of minimum error is defined by (i, j) such that $D(i, j)$ is minimum [20]. The 2D logarithmic search follows the direction of minimum error. Five search points are checked in each step, as shown in Fig. 5. The distance between the search points is reduced if the minimum is in the center of the search locations or at the boundary of search area. In the example shown, seven steps are required to find the motion vector at (4, 7).

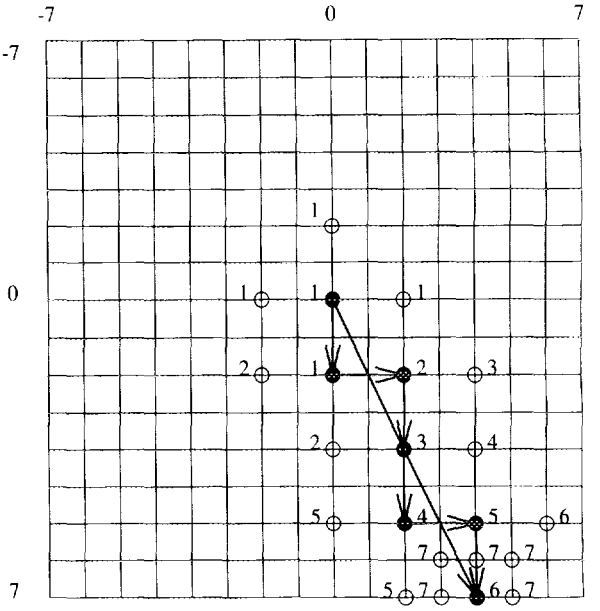

FIG. 5. The 2D logarithmic search procedure [16].

The motion estimation is performed only on the luminance samples. The chrominance displacement is approximated by halving the luminance displacement. In order to have further improved accuracy in prediction, after doing an integral full-pixel search, a half-pel search is also done on a neighborhood of eight bilinearly interpolated luminance samples from the reconstructed reference frame.

In our implementation, as described in Section 4.1 and Section 4.2, the data distribution is done such that each processor has all the data that it needs to conduct motion estimation within the window size. As a result, all processors can concurrently generate their motions vectors.

4.3.3. Rate Control and Adaptive Quantization

The process of uniform quantization of the 8 × 8 DCT coefficients may be expressed as

$$Cq_{i,j} = \text{int}\left[\frac{C_{i,j}}{Q_{i,j}} + \frac{k}{2}\right], \quad i, j = 1, ..., 8, \qquad (8)$$

where the resulting quantized coefficients $Cq_{i,j}$ is the integer part of the bracketed term, $C_{i,j}$ being the DCT coefficients and $Q_{i,j}$ the corresponding quantizer steps; while the parameter k takes on values 1 or 0, for quantization with rounding to nearest integer and for quantization with truncation, respectively [9]. (Note that, this formula as-

sumes $C_{i,j}$ is positive, however, it is trivial to generalize this formula to take care of negative DCT coefficients).

For MPEG-2, $Q_{i,j} = q_p\omega_{i,j}/8$, where $\omega_{i,j}$ is a matrix of quantization weights and q_p is the quantizer scaling factor. There may be two integer matrices of $\omega_{i,j}$ to code a sequence. Adaptive quantization results from the variability of q_p which is allowed to vary between 1 and 31 on a macroblock-to-macroblock basis. The number of bits allocated for quantized data is dependent on the q_p. Therefore, the value of q_p can be adjusted according to the bit rate requirements.

Our implementation of the encoder holds fast a single pass coding viewpoint and does not use any a priori measurement to guide the allocation of bits at the global layers. The complex bit allocation process is split into a number of independent stages, coincident with the various layers of MPEG-2 video. At the highest stage, a Group of Pictures (*GOP*) becomes the edge where variable size coded pictures are mapped into a constant channel rate. The *GOP* is defined to be the distance between successive I pictures, but it also appears as a convenient division for rate control [8].

The allocation of target bit for the current picture being encoded is based on a global bit budget for the *GOP*, that is, a bit budget for the coded sequence of pictures, and a ratio of weighted relative coding complexities of the three picture types. Coding complexity is estimated in each processor as the product of the average macroblock quantization stepsize and the number of bits generated by each processor by coding its part of the frame. The local bit allocation for the current macroblock is based on two measurements: first, the deviance from estimated buffer fullness for the current (jth) macroblock and second, the normalized spatial activity. The estimated buffer fullness is simply the product of the macroblock number and the average bits per macroblock. The spatial activity is a measure which performs the second adjustment to the quantization step size. We have chosen variance of the macroblock in question as this measure, which is computed a priori on the four source luminance blocks (8×8) regardless of the ultimate macroblock prediction mode (in practice, variance has been computed concurrent with the motion estimation stage). The variance measure is then normalized against the average variance of the most recently coded picture. The picture-fragment in each processor is approximated and estimated to have a uniform distribution of bits. If the local trend of generated bits begins to stray from this estimation, a compensation factor emerges to modulate the macroblock quantization scale. The compensation factor is the difference between the predicted and the true buffer fullness of the jth macroblock.

Since the allocation of bit depends on the global bit budget, the quantization and rate control process is inherently sequential in each processor. The global bit budget is broadcasted to all the processors to perform the allocation of target bit.

5. EXPERIMENTAL RESULTS

Experiments were performed on the Intel Paragon using various number of processors. The measured time was averaged over 50 frames of a video sequence. We used five video sequences: Football (360×240), Table Tennis (360×240), Salesman (360×288), Miss America (352×288), and Swing (352×288). The Football sequence involves a football game, where players move in fast motion. The Table Tennis sequence is a bouncing pingpong ball with two players playing the ball; it involves camera panning and zooming and a scene change. The Salesman sequence shows a person holding an object while talking; the salesman moves his hands and the object rapidly. The Miss America sequence is basically a head and shoulder sequence, and the motion involves a very large area of the frame. The Swing sequence consists of a cluster of different charts and graphs, which also involves some movements. All of these sequences are representative of different kinds of motion and are very useful for testing motion estimation.

The time to process 50 frames of a video sequence was not necessarily the same in each processor, so the average was taken over all the processors. Several such sets of measurements were taken using 1, 2, 4, 8, 16, 32, 64, 128, 256, and 330 processors for each set. All of the timings were measured with microsecond granularity.

We used a constant bit-rate of 5 Mbp and a *video-buffering-verifier* buffer size of 112 as input for all 50 frames, with a GOP of 12 and an I-to-P frame distance of 3, while the search window was ± 11 pels for P-pictures and ± 10 pels for B-pictures. Both full search and 2D logarithmic

TABLE II
Permitted Limits of Parameters of MPEG-2[a]

Name of Sequence	Average PSNR: Full Search (dB)	Average PSNR: 2D Log Search (dB)
Football	37.1060	34.5281
Table Tennis	38.5159	35.7497
Salesman	39.8703	34.5039
Miss America	42.2098	37.1647
Swing	41.4682	37.3657

search were performed and the corresponding performance was monitored. In order to measure the quality of the video, we used the peak signal-to-noise ratio (*PSNR*) as there exists no good and simple metric for this measure [10]. The PSNR of a video is defined as

$$PSNR = 10 \log_{10} \frac{255 \times 255}{MSE}, \quad (9)$$

where *MSE* is the mean square error. The larger the *PSNR*, the better the quality. The average values of *PSNR* obtained for different sequences are shown in Table II.

In order to speed up the motion estimation process, the search pattern was changed to outward spiralling order in place of initial line scan order, and the inner loop of the block distance calculation was unrolled. Furthermore, while computing the distance, accumulation was stopped if the partial sum exceeded a given distance. Figures 6 through 10 show various results for the encoder employing full-search motion estimation. Figure 6 depicts the speedup of the motion estimation module which is the most computationally expensive part. It is clear that the problem scaled almost linearly as the number of processor was increased. Figure 7 shows the speedup of the DCT module which also increased at a near-linear rate.

Figure 8 gives the overall speedup with full search motion estimation algorithm. As can be noticed, the speedup curves for various video sequences are almost linear up to 128 processors and then start saturating. Figure 9 shows a comparison of the computational modules in terms of the time required by each module. This bar graph indicates where the bottleneck of the encoding is. In this case, it is obvious that motion estimation was the most time-consuming module. On the average, using optimizations, full-search motion estimation took 71.5% of the total encoding time. With optimizations, this figure reduced to 53.43% but the motion estimation still remained the most time-consuming module of the entire encoding process. Figure 10 depicts the average encoding speed of the entire encoding process; the peak rate (averaged across all sequences) was 11.76 frames/s with 330 processors. Clearly, this frame

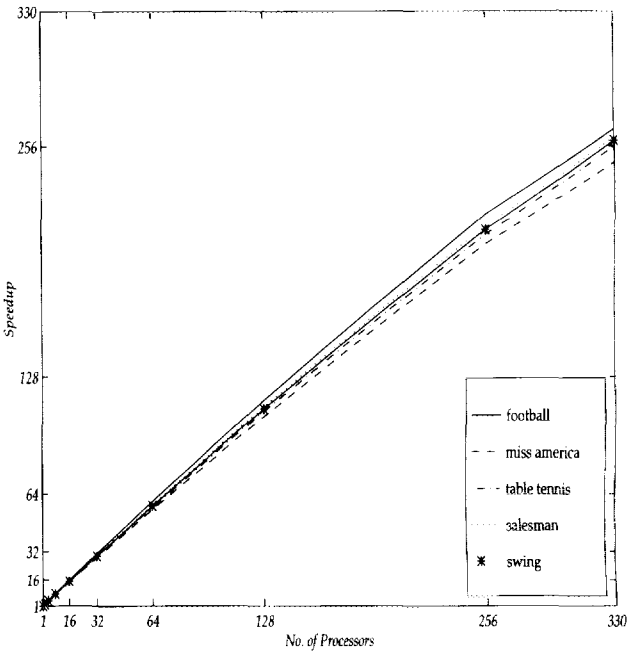

FIG. 7. Speedup for DCT: full search.

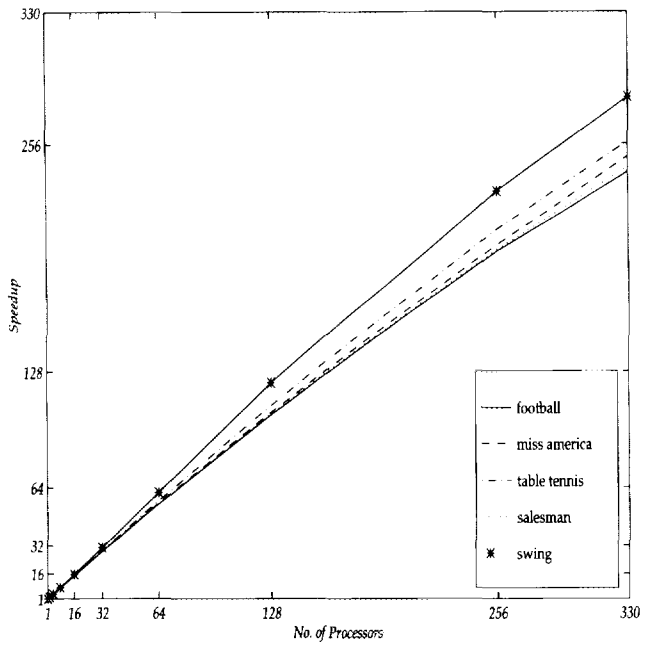

FIG. 6. Speedup for motion estimation: full search.

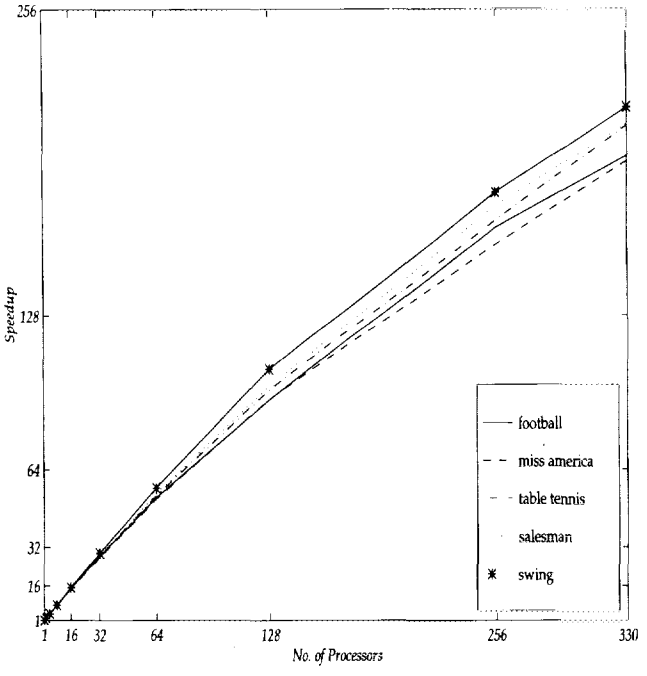

FIG. 8. Overall speedup with full-search motion estimation.

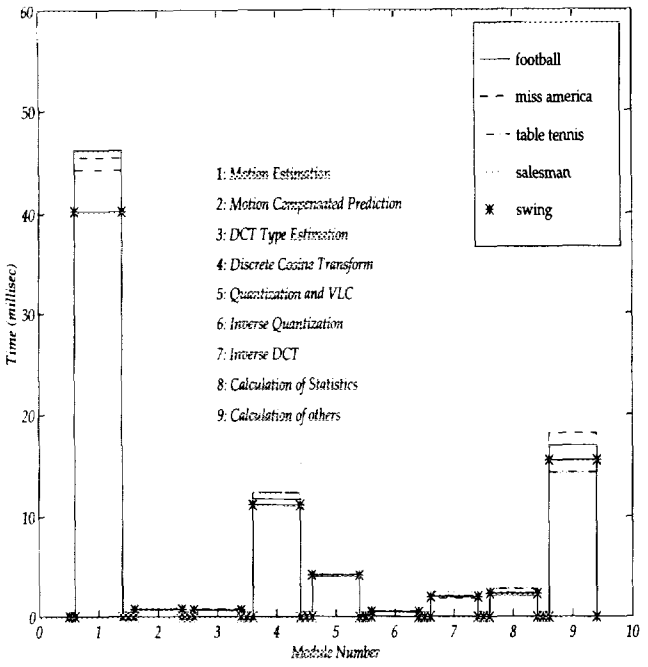

FIG. 9. Comparison of modules using 128 processors: full search.

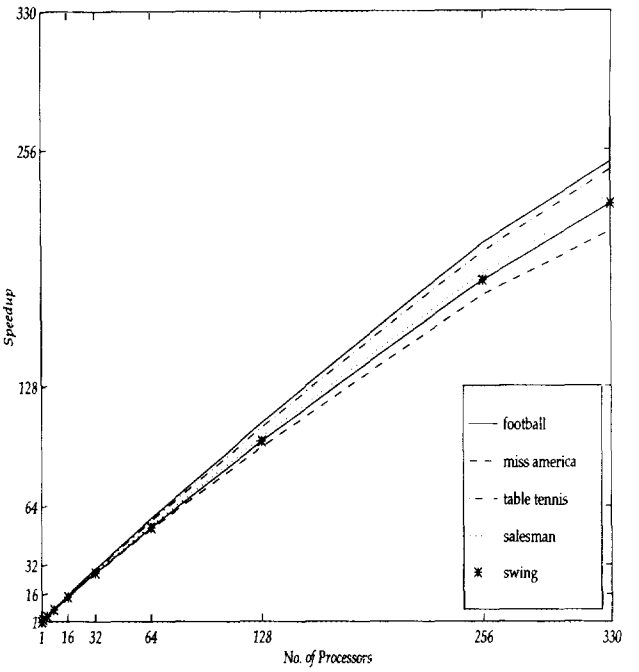

FIG. 11. Speedup for motion estimation: logarithmic search.

rate is not very impressive, though a high speedup in computation was achieved. To obtain a faster encoding rate, we employed a the *logarithmic search.*

Figures 11 to 15 show the results obtained when motion estimation was done with the logarithmic search and DCT was done with Wang's algorithm. First, we examine the speedup for the motion estimation module which is given in Fig. 11. The speedup was close to that of full search, but the time required to do the search was decreased remarkably. This is because the number of points searched is much smaller with the logarithmic search as compared to the full search. This provided roughly a ninefold increment in the motion estimation and a threefold increment in the overall encoding speed. Figure 12 shows the speedup for DCT using Wang's algorithm indicating that fast DCT routine also scaled almost the same way as the direct approach (given in Fig. 7).

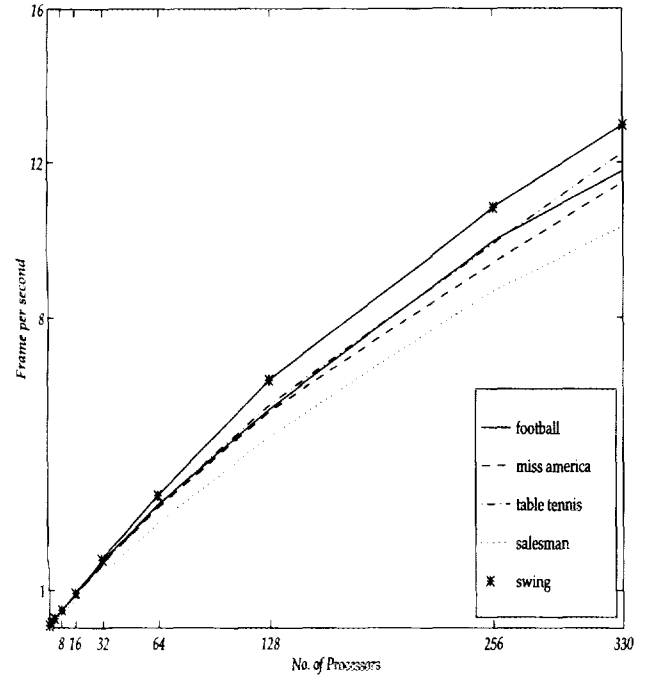

FIG. 10. Encoding speed: full search.

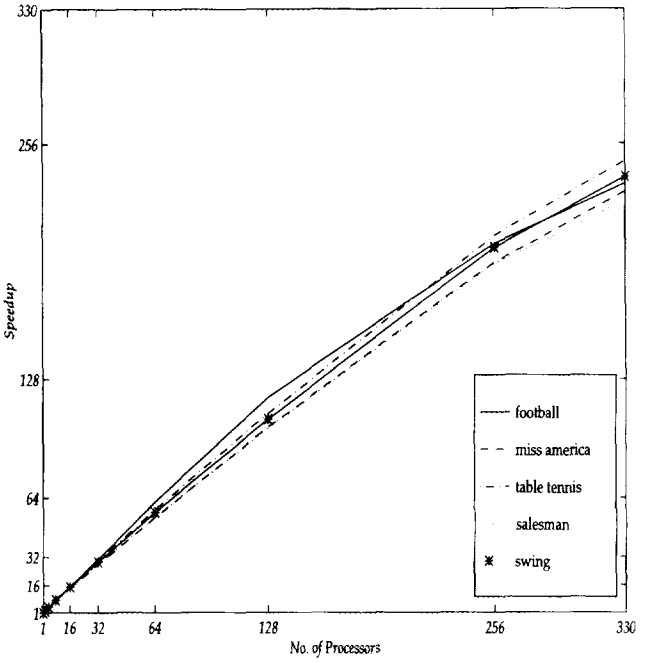

FIG. 12. Speedup for fast DCT while using logarithmic search.

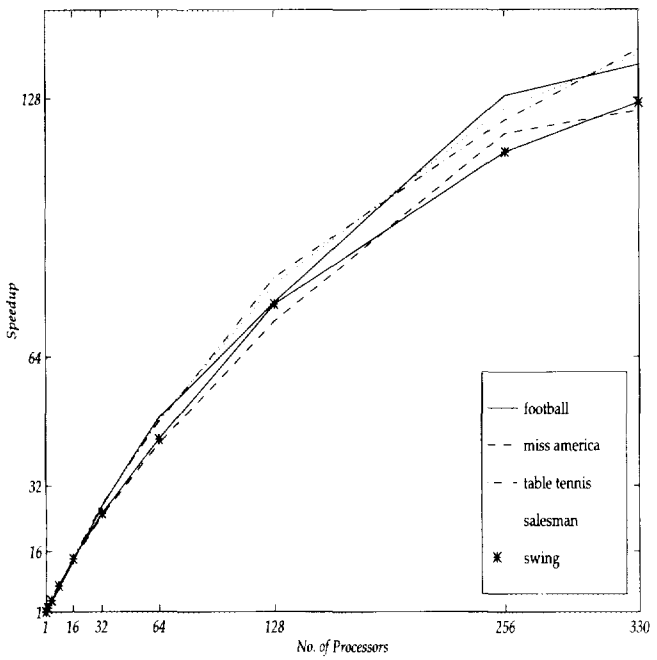

FIG. 13. Overall speedup with logarithmic-search motion estimation.

Figure 13 gives the overall speedup using the logarithmic search with various number of processors. It is noticeable that the overall speedup was less as compared to the full-search realization: it can be observed that the overall speedup was close to 150 for the full-search implementation, and about 130 for the logarithmic search implementation. This is because motion estimation did not dominate anymore, rather the "others" module, which included various housekeeping functions and some constant overhead, became the predominant factor. This module did not exhibit very regular behavior and, in fact, gave a speedup less than 1. There was a rapid increase in the speedup up to 128 processors. By increasing the number of processors from 128 to 256, an escalation in speedup was still observed. However, a little improvement was obtained beyond 256 processors. The reason is that the times for all the modules except the "others" module approached their minimum limits.

Figure 14 provides a comparison among the times for various modules. It can be noticed that the relative processing time required for motion estimation was considerably smaller. The motion estimation module now took only 18.1% of the total encoding time. As can be seen from Fig. 15, the frame processing rate became about three times faster than before. The maximum encoding rate was achieved for the Swing sequence (31.14 frames/s). The peak rate, averaged over all the sequences, was 29.36 frames/s.

Tables III through VII show the timings of the computational modules for all the sequences using various number of processors for the full-search implementation, while Tables VIII through XII show the timings of the computational modules for all the sequences using various number of processors for the logarithmic search implementation.

6. CONCLUSIONS

In this paper, an efficient parallel implementation of the MPEG-2 encoder was described. The strategies for data distribution were discussed. The implementation was per-

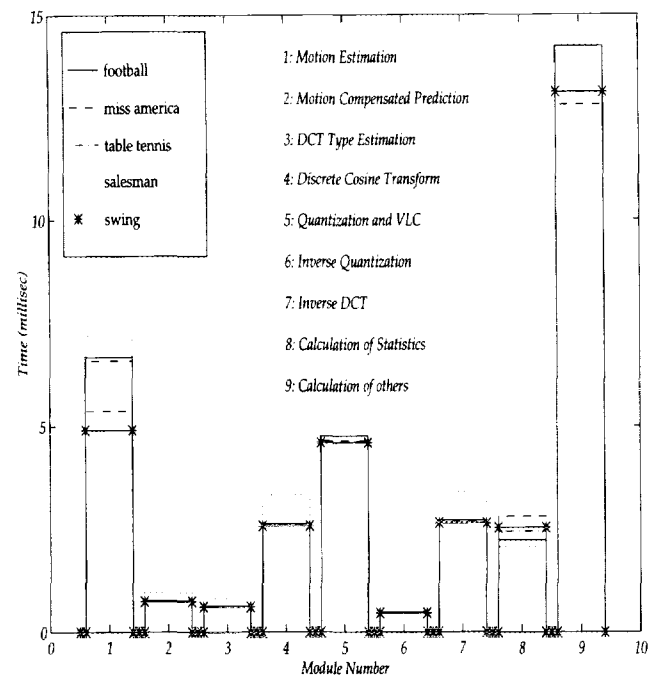

FIG. 14. Comparison of modules using 330 processors: logarithmic search.

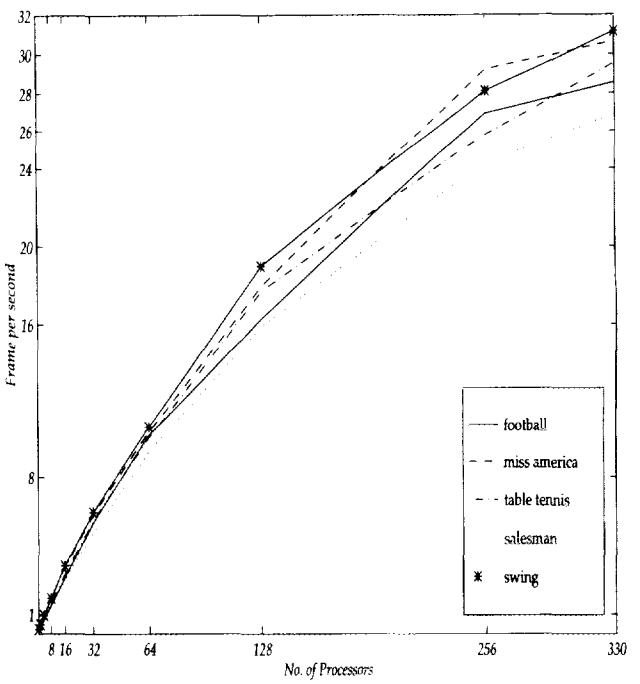

FIG. 15. Encoding speed: logarithmic search.

TABLE III
Timings (ms) for Football Sequence: Full Search

Name of module	Number of processors									
	1	2	4	8	16	32	64	128	256	330
Motion Estimation	11113.62	5599.37	2840.59	1452.10	746.89	386.75	202.34	106.88	56.56	46.25
Motion-Comp. Prediction	208.89	104.96	52.89	26.71	13.51	6.84	3.47	1.76	0.89	0.73
DCT Type Estimation	189.38	95.09	47.99	24.31	12.34	6.28	3.20	1.63	0.83	0.68
Discrete Cosine Transform	3104.99	1553.35	778.08	390.00	195.53	100.10	51.81	26.97	14.20	11.71
Quantization and VLC	978.18	494.33	251.15	128.71	66.23	34.22	17.81	9.32	4.90	4.02
Inverse Quantization	139.70	70.29	35.32	17.78	8.96	4.52	2.28	1.15	0.58	0.48
Inverse DCT	501.31	252.27	127.72	65.03	33.30	17.14	8.88	4.61	2.39	1.99
Calculation of Statistics	235.69	117.98	59.28	30.39	16.13	9.08	5.57	3.51	2.54	2.10
Calculation of Others	25.52	15.57	18.25	15.37	10.87	17.87	19.33	22.34	17.21	16.97
Total	16497.28	8303.21	4211.27	2150.40	1103.76	582.41	314.69	178.17	100.10	84.93

TABLE IV
Timings (ms) for Table Tennis Sequence: Full Search

Name of module	Number of Processors									
	1	2	4	8	16	32	64	128	256	330
Motion Estimation	11397.22	5731.28	2898.54	1475.31	754.56	388.31	200.70	104.31	54.58	44.34
Motion-Comp. Prediction	182.67	93.00	47.53	24.41	12.50	6.43	3.32	1.73	0.91	0.75
DCT Type Estimation	180.80	91.22	46.11	23.45	11.99	6.17	3.21	1.67	0.88	0.72
Discrete Cosine Transform	3162.57	1593.72	805.56	409.50	209.67	107.77	55.81	29.01	15.26	12.36
Quantization and VLC	1064.72	538.09	273.45	139.22	71.28	36.70	19.00	9.89	5.20	4.19
Inverse Quantization	130.32	65.56	33.21	16.87	8.60	4.41	2.28	1.19	0.63	0.52
Inverse DCT	532.58	268.37	135.64	68.91	35.22	18.11	9.37	4.88	2.57	2.09
Calculation of Statistics	234.89	117.33	59.39	30.36	16.10	8.97	5.63	3.52	2.56	2.39
Calculation of Others	13.04	11.50	17.55	12.27	14.29	17.49	18.00	18.51	18.13	14.34
Total	16898.81	8510.07	4316.98	2200.30	1134.21	594.36	317.32	174.71	100.72	81.70

TABLE V
Timings (ms) for Salesman Sequence: Full Search

Name of module	Number of Processors									
	1	2	4	8	16	32	64	128	256	330
Motion Estimation	13423.19	6784.53	3466.27	1771.76	911.73	469.19	243.94	128.30	67.96	55.20
Motion-Comp. Prediction	224.60	113.13	57.08	28.96	14.74	7.54	3.88	2.02	1.06	0.86
DCT Type Estimation	195.34	98.38	49.86	25.38	12.98	6.67	3.45	1.80	0.95	0.78
Discrete Cosine Transform	3994.16	2009.24	1015.90	513.99	261.48	133.80	68.86	35.79	18.85	15.31
Quantization and VLC	1087.45	547.12	276.66	140.29	71.42	36.51	18.77	9.78	5.15	4.21
Inverse Quantization	124.56	62.53	31.48	15.91	8.09	4.16	2.15	1.12	0.59	0.48
Inverse DCT	646.87	324.72	163.36	82.33	41.68	21.17	10.84	5.63	2.96	2.43
Calculation of Statistics	279.58	139.90	70.44	36.08	18.98	10.50	6.30	3.97	3.21	2.82
Calculation of Others	12.55	14.53	14.55	18.51	18.96	16.20	11.09	15.21	14.25	14.26
Total	19988.30	10094.08	5145.60	2633.21	1360.06	705.74	369.28	203.62	114.98	96.35

TABLE VI
Timings (ms) for Miss America Sequence: Full Search

Name of module	Number of Processors									
	1	2	4	8	16	32	64	128	256	330
Motion Estimation	11332.69	5701.41	2901.78	1482.32	760.48	392.75	204.41	107.45	56.62	45.55
Motion-Comp. Prediction	185.97	94.55	48.07	24.58	12.59	6.48	3.37	1.76	0.93	0.76
DCT Type Estimation	198.13	99.95	50.54	25.73	13.16	6.78	3.52	1.84	0.97	0.80
Discrete Cosine Transform	3061.24	1542.42	783.99	401.10	206.54	106.77	55.28	28.83	15.16	12.38
Quantization and VLC	1078.29	542.32	274.29	139.42	71.33	36.67	18.97	9.92	5.23	4.25
Inverse Quantization	131.91	66.47	33.70	17.18	8.81	4.54	2.35	1.23	0.65	0.54
Inverse DCT	446.93	225.75	114.70	58.63	30.03	15.47	8.00	4.16	2.19	1.83
Calculation of Statistics	271.67	136.91	68.94	35.33	18.53	10.36	6.21	3.87	2.98	2.76
Calculation of Others	13.00	11.58	18.70	15.35	22.20	18.97	20.25	20.99	21.16	18.17
Total	16719.83	8421.36	4294.71	2199.64	1143.67	598.79	322.36	180.05	105.89	87.04

TABLE VII
Timings (ms) for Swing Sequence: Full Search

Name of module	Number of Processors									
	1	2	4	8	16	32	64	128	256	330
Motion Estimation	11332.58	5691.34	2861.85	1441.59	726.36	366.41	184.95	93.07	49.30	40.24
Motion-Comp. Prediction	182.10	92.07	46.61	23.72	12.13	6.21	3.21	1.67	0.88	0.72
DCT Type Estimation	177.43	89.02	44.79	22.65	11.49	5.85	3.00	1.56	0.82	0.72
Discrete Cosine Transform	2885.64	1455.85	737.29	374.30	190.67	97.79	50.32	26.12	13.74	11.14
Quantization and VLC	1075.26	539.98	272.39	137.79	70.06	35.85	18.45	9.65	5.08	4.16
Inverse Quantization	123.19	61.91	31.20	15.78	8.02	4.10	2.11	1.10	0.58	0.47
Inverse DCT	508.28	255.43	128.70	65.12	33.15	16.97	8.78	4.57	2.41	1.98
Calculation of Statistics	237.07	119.63	60.73	31.43	16.07	8.97	5.53	3.71	2.61	2.30
Calculation of Others	16.23	14.91	12.49	17.09	15.56	16.60	16.55	15.66	16.84	15.53
Total	16537.78	8320.14	4196.05	2129.47	1083.51	558.75	292.90	157.11	92.26	77.21

TABLE VIII
Timings (ms) for Football Sequence: Logarithmic Search

Name of module	Number of Processors									
	1	2	4	8	16	32	64	128	256	330
Motion Estimation	1668.48	841.36	423.13	213.82	108.70	55.94	29.12	15.32	8.12	6.69
Motion-Comp. Prediction	164.00	81.82	43.12	21.80	10.82	5.35	2.76	1.46	0.87	0.76
DCT Type Estimation	149.19	74.65	38.96	19.52	9.75	4.90	2.44	1.27	0.73	0.63
Discrete Cosine Transform	606.98	303.42	158.69	79.38	39.80	19.76	9.86	5.14	3.00	2.59
Quantization and VLC	1176.28	762.08	462.44	226.01	105.61	43.92	21.68	11.27	5.84	4.76
Inverse Quantization	124.78	62.61	31.64	16.02	8.15	4.17	2.15	1.11	0.58	0.48
Inverse DCT	632.06	316.09	165.07	82.58	41.25	20.59	10.36	5.34	3.10	2.67
Calculation of Statistics	235.65	117.95	59.32	30.43	16.15	9.11	5.58	3.52	2.55	2.20
Calculation of Others	17.31	18.60	10.99	15.00	10.30	12.82	14.14	17.04	12.34	14.24
Total	4774.73	2579.58	1393.36	784.56	350.53	176.56	98.09	61.47	37.14	35.02

TABLE IX
Timings (ms) for Table Tennis Sequence: Logarithmic Search

Name of module	Number of Processors									
	1	2	4	8	16	32	64	128	256	330
Motion Estimation	1620.74	816.62	413.86	210.65	107.62	55.65	28.96	15.23	8.07	6.61
Motion-Comp. Prediction	184.98	92.81	46.67	23.72	12.16	6.29	3.27	1.71	0.91	0.76
DCT Type Estimation	153.66	77.27	39.27	19.88	10.12	5.20	2.69	1.41	0.75	0.63
Discrete Cosine Transform	625.76	314.25	158.18	79.95	40.68	20.90	10.88	5.72	3.03	2.54
Quantization and VLC	1131.88	569.10	288.68	147.11	75.45	38.96	20.28	10.59	5.58	4.64
Inverse Quantization	117.37	58.85	29.61	15.04	7.72	3.98	2.07	1.09	0.58	0.48
Inverse DCT	658.77	330.48	166.35	84.07	42.73	21.82	11.28	5.89	3.14	2.62
Calculation of Statistics	234.07	117.47	59.40	30.44	16.16	8.99	5.64	3.53	2.56	2.41
Calculation of Others	15.34	12.37	10.52	16.66	16.22	11.16	14.47	11.41	14.16	13.14
Total	4742.57	2389.22	2254.02	627.52	328.86	172.95	99.54	56.58	38.78	33.83

TABLE X
Timings (ms) for Salesman Sequence: Logarithmic Search

Name of module	Number of Processors									
	1	2	4	8	16	32	64	128	256	330
Motion Estimation	1672.48	840.42	425.25	218.62	113.68	59.39	31.22	16.52	8.86	7.21
Motion-Comp. Prediction	222.59	112.22	56.87	28.90	14.81	7.66	4.01	2.11	1.12	0.95
DCT Type Estimation	177.82	90.18	45.89	23.66	12.25	6.37	3.35	1.77	0.95	0.81
Discrete Cosine Transform	740.07	373.74	189.99	97.68	50.61	26.44	13.84	7.30	3.87	3.29
Quantization and VLC	1153.79	579.10	292.64	148.47	75.47	38.70	19.95	10.49	5.55	4.76
Inverse Quantization	112.10	56.53	28.66	14.57	7.48	3.88	2.02	1.07	0.57	0.49
Inverse DCT	792.47	399.31	201.92	102.18	52.49	27.07	14.10	7.45	3.97	3.36
Calculation of Statistics	280.01	139.25	70.61	36.03	19.03	10.53	6.30	3.88	2.41	2.04
Calculation of Others	12.74	11.88	17.62	19.80	14.76	13.39	12.30	12.37	13.91	14.26
Total	5164.07	2602.63	1329.45	689.91	360.58	193.43	107.09	62.96	41.21	37.17

TABLE XI
Timings (ms) for Miss America Sequence: Logarithmic Search

Name of module	Number of Processors									
	1	2	4	8	16	32	64	128	256	330
Motion Estimation	1140.79	574.29	295.58	152.92	79.60	41.86	22.11	11.89	6.45	5.38
Motion-Comp. Prediction	188.36	94.65	47.70	24.13	12.28	6.31	3.27	1.71	0.90	0.75
DCT Type Estimation	140.84	71.25	36.50	18.75	9.69	5.04	2.64	1.39	0.74	0.62
Discrete Cosine Transform	589.07	301.04	154.58	79.58	41.04	21.23	11.05	5.78	3.07	2.56
Quantization and VLC	1014.21	515.59	263.68	135.69	70.22	36.62	19.19	10.14	5.40	4.62
Inverse Quantization	106.35	54.36	27.94	14.41	7.45	3.88	2.03	1.07	0.57	0.47
Inverse DCT	604.91	310.18	159.38	81.91	42.37	21.96	11.43	5.98	3.15	2.63
Calculation of Statistics	273.05	136.34	69.10	35.26	18.66	10.41	6.24	3.88	2.88	2.77
Calculation of Others	12.34	14.22	17.76	11.13	12.80	17.52	18.49	13.95	11.05	12.83
Total	4069.92	2071.92	1072.22	553.78	294.11	164.83	96.45	55.79	34.21	32.63

TABLE XII
Timings (ms) for Swing Sequence: Logarithmic Search

Name of module	Number of Processors									
	1	2	4	8	16	32	64	128	256	330
Motion Estimation	1114.07	566.44	289.43	148.90	77.30	40.43	21.26	11.24	6.02	4.91
Motion-Comp. Prediction	178.22	89.67	45.55	23.28	11.94	6.16	3.20	1.67	0.88	0.74
DCT Type Estimation	149.16	75.54	38.45	19.61	10.03	5.17	2.68	1.40	0.74	0.61
Discrete Cosine Transform	604.16	303.80	153.70	78.15	40.19	20.77	10.80	5.69	3.02	2.54
Quantization and VLC	1012.33	513.98	262.88	135.36	70.02	36.59	19.24	10.21	5.48	4.59
Inverse Quantization	109.16	54.92	27.93	14.28	7.29	3.76	1.95	1.02	0.54	0.46
Inverse DCT	617.87	310.66	157.60	80.16	41.04	21.31	11.12	5.85	3.10	2.61
Calculation of Statistics	273.00	137.04	69.11	35.26	18.57	10.40	6.24	3.88	2.76	2.50
Calculation of Others	12.36	16.15	11.04	12.33	10.52	16.77	17.56	11.79	13.04	13.15
Total	4070.33	2068.20	1055.69	547.33	286.90	161.36	94.05	52.75	35.58	32.11

formed on an MIMD machine using the SPMD programming model. Exploiting its advantages, different MPEG-2 modules were parallelized. Noticeable improvements in speedup were achieved for the individual modules as well as the overall speedup of the MPEG-2 encoder. In our implementation, the I/O was not handled by dedicated processors, otherwise further improvement in speedup is expected. We used full search and 2D logarithmic search algorithms for motion estimation but our implementation allows inclusion of faster algorithms which can further reduce the total computation time. Motion estimation techniques have been reported in the literature, for example, [7], which have underlying parallelism.

Our implementation is scalable allowing the control of granularity according to the available hardware. It is to be noted, however, that the number of processors cannot be increased indefinitely. Theoretically, this number is upper bounded by the number of macroblocks in a frame. In our experiments conducted on the Intel Paragon XP/S, we have used this upper bound which is 330 processors. We have achieved a maximum frame rate of 31.14 frames/s, which is higher than real-time requirements.

The current implementation is done with Express message-passing system allowing portability. The results of our experiments and comparisons on various other architectures have been reported elsewhere [2]. Current efforts, in addition to investigating various code optimization strategies, are directed towards testing diverse motion estimation algorithms and using other message-passing libraries such as PVM and MPI.

ACKNOWLEDGMENTS

We thank Samuel Kwan of the Center for Computing Services and Telecommunications at HKUST and Adam Kolawa and Arthur Hicken of Parasoft Corporation for their assistance in running our programs on the Paragons.

REFERENCES

1. T. Akiyama, H. Aono, K. Aoki, K. W. Ler, B. Wilson, T. Araki, T. Morishige, H. Takeno, A. Sato, S. Nakatani, and T. Senoh, MPEG2 video codec using image compression DSP, *IEEE Trans. Consumer Electron.* **40,** 3 (Aug. 1994), 466–472.
2. S. M. Akramullah, Real-time MPEG-2 video encoding on parallel and distributed systems, M.Phil. Thesis, The Hong Kong University of Science and Technology, June 1995.
3. R. Aravind, G. L. Cash, D. L. Duttweiler, H. M. Hang, B. G. Haskell, and A. Puri, Image and video coding standards, *AT&T Tech. J.* (Jan.–Feb. 1993), 67–89.
4. T. Bonniger, R. Esser, and D. Krekel, CM-5E, KSR2, Paragon XP/S: A comparative description of massively parallel computers, *Parallel Comput.* **21** (1995), 199–232.

5. CCITT, Recommendation H.261—Video codec for audiovisual services at $p \times 64$ kbits/s, Geneva, Aug. 1990.
6. N. I. Cho and S. U. Lee, Fast algorithm and implementation of 2-D discrete cosine transform, *IEEE Trans. Circuits and Systems* **38,** 3 (Mar. 1991), 297–305.
7. K. H. K. Chow and M. L. Liou, Genetic motion search algorithm for video compression, *IEEE Trans. Circuits and Systems for Video Tech.* **3,** 6 (Dec. 1993), 440–445.
8. C. Fogg, P. Au, S. Eckart, T. Hanamura, K. Oosa, B. Quandt, and H. Watanabe, ISO/IEC software implementation of MPEG-1 Video, *Proceedings of the SPIE, Conference on Digital Video Compression on Personal Computers: Algorithms and Technologies,* San Jose, CA, Feb. 6–10, 1994, Vol. 2187, pp. 249–257.
9. C. A. Gonzales and E. Viscito, Motion video adaptive quantization in the transform domain, *IEEE Trans. Circuits and Systems for Video Tech.* **1,** 4 (Dec. 1991), 374–378.
10. K. L. Gong and L. A. Rowe, Parallel MPEG-1 video encoding, *1994 Picture Coding Symposium,* Sacramento, CA, Sep. 1994.
11. Z. Huang, Y. Takeuchi, and H. Kunieda, Distributed load balancing schemes for parallel video encoding system, *IEICE Trans. Fundam. Electron. Comm. Comput. Sci.* **E77-A,** 5 (May 1994), 923–930.
12. Intel Supercomputer Systems Division, Intel Corporation, Intel Paragon XP/S technical summary, Jan. 1994.
13. ISO Committee Draft 13818-2, Generic coding of moving pictures and associated audio: Recommendation H.262, ISO/IEC JTC1/SC29 WG11/602, Seoul, Nov. 1993.
14. ISO Committee Draft 10918-1, Digital compression and coding of continuous-tone still images—Part 1: Requirements and guidelines, ISO/IEC DIS 10918-1, Feb. 1991.
15. ISO Committee Draft 11172-2, Coding of moving pictures and associated audio for digital storage media at upto 1.5 Mbits/s, ISO/IEC JTC1/SC29 WG11, Nov. 1991.
16. J. R. Jain and A. K. Jain, Displacement measurement and its application in interframe image coding, *IEEE Trans. Comm.* **COM-29,** 12 (Dec. 1981), 1799–1808.
17. D. J. LeGall, MPEG: A video compression standard for multimedia applications, *Comm. ACM* **34,** 4 (Apr. 1991), 46–58.
18. 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,* Raleigh, NC, Sep. 2–3, 1992, pp. 80–85.
19. *MPEG-2 Video Encoder,* Version 1.1a, MPEG Software Simulation Group, July 1994.
20. H. G. Musmann, P. Pirsch, and H. J. Graller, Advances in picture coding, *Proc. IEEE* **73,** 4 (Apr. 1985), 523–548.
21. Parasoft Corp., *Express* system user's guide, Parasoft Corporation, Pasadena, CA, 1992.
22. P. Pirsch and H. Jeschke, A MIMD multiprocessor system for real-time image processing, *Proceedings of the SPIE, Image Processing Algorithms and Techniques II,* 1991, Vol. 1452, pp. 544–555.
23. J. Poskanzer, *PBMPLUS: Extended Portable Bitmap Toolkit,* Dec. 10, 1991.
24. K. Shen, L. A. Rowe, and E. J. Delp, A parallel implementation of an MPEG1 encoder: Faster than real-time!, *Proceedings of the SPIE, Conference on Digital Video Compression: Algorithms and Technologies,* San Jose, CA, Feb. 1995.
25. F. Sijstermans and J. van der Meer, CD-I full-motion video encoding on a parallel computer, *Comm. ACM* **34,** 4 (Apr. 1991), 81–91.
26. Y. Takeuchi, Z. C. Huang, M. Saeki, and H. Kunieda, RHINE: Reconfigurable multiprocessor system for video codec, *IEICE Trans. Fundam. Electron. Comm. Comput. Sci.* **E76-A,** 6 (June 1993), 947–956.
27. H. H. Taylor, D. Chin, and A. W. Jessup, An MPEG encoder implementation on the princeton engine video supercomputer, *Data Compression Conference 1993,* IEEE Comput. Soc. Press, Los Alamitos, CA, 1993, pp. 420–429.
28. Z. Wang, Fast algorithms for the discrete W transform and for the discrete Fourier transform, *IEEE Trans. Acoust. Speech Signal Process.* **ASSP-32,** 4 (Aug. 1984), 803–816.
29. Y. Yu and D. Anastassiou, Software implementation of MPEG-II video encoding using socket programming in LAN, *Proceedings of the SPIE, Conference on Digital Video Compression on Personal Computers: Algorithms and Technologies,* San Jose, CA, Feb. 6–10, 1994, Vol. 2187, pp. 229–240.

Received October 5, 1994; revised April 11, 1995; accepted May 26, 1995

SHAHRIAR MOHAMMAD AKRAMULLAH received his B.Sc. degree in electrical and electronic engineering from the Bangladesh Institute of Technology, Chittagong, Bangladesh in 1991. Currently, he is a Ph.D. student at the Hong Kong University of Science and Technology in the Department of Electrical and Electronic Engineering. He was with the Department of Electrical and Electronic Engineering at BIT, Chittagong, Bangladesh from 1991 to 1993. He is a recipient of the Commonwealth Scholarship and Fellowship Plan (Hong Kong Awards). His research interests include video and multimedia technology, parallel processing, and distributed algorithms for real-time video and image processing. He is a student member of the IEEE and an associate member of the Institution of Engineers, Bangladesh.

ISHFAQ AHMAD received his B.Sc. degree in electrical engineering from the University of Engineering and Technology, Lahore, Pakistan, and his M.S. degree in computer engineering and Ph.D. degree in computer science from Syracuse University. His research interests include various aspects of parallel and distributed computing, high-performance computer architectures and their assessment, and performance evaluation. Currently, he is a faculty member in the Department of Computer Science at the Hong Kong University of Science and Technology. He received Best Student Paper Awards at Supercomputing '90 and Supercomputing '91. He has been a guest editor for a special issue of *Concurrency: Practice and Experience.* Dr. Ahmad is a member of the IEEE Computer Society.

MING L. LIOU received his B.S. degree from National Taiwan University, his M.S. degree from Drexel University, Philadelphia, PA, and his Ph.D. degree from Stanford University, Stanford, CA, in 1956, 1961, and 1964, respectively, all in electrical engineering. He joined the faculty of the Department of Electrical and Electronic Engineering, the Hong Kong University of Science and Technology, as a professor in October 1992 and was appointed as the Director of Hongkong Telecom Institute of Information Technology in January 1993. His current research interests include very low bit-rate video, motion estimation techniques, packet video, HDTV, VLSI architecture, implementation of signal processing systems for visual applications, and information technology. From 1984 to 1992, he was a director at Bellcore, Red Bank, NJ, conducting research in data transmission, digital subscriber line transceiver, and video technology. He joined AT&T Bell Labs in 1963 as a member of the technical staff and had held various supervisory positions until 1984 when he was transferred to Bellcore. During his career at AT&T Bell Labs, he did research on numerical analysis, system theory, FM distortion analysis, and computer-aided design of communication circuits and systems, including circuits containing periodically operated switches. Professor Liou has published numerous papers in various fields and received the IEEE Circuits and Systems Society Special Prize Paper Award in 1973 and the Darlington Prize Paper Award in 1977. He has been very active in professional activities and served in various capacities including Editor of the *IEEE Transactions on Circuits and Systems* from 1979 to 1981, President of the IEEE Circuits and Systems Society, and the Founding Editor of the *IEEE Transactions on Circuits and Systems for Video Technology* since 1991. He is a member of Sigma Xi, Eta Kappa Nu, Phi Tau Phi, and a fellow of the IEEE and the Hong Kong Institution of Engineers.