Transactions Briefs

Performance of Software-Based MPEG-2 Video Encoder on Parallel and Distributed Systems

Shahriar M. Akramullah, Ishfaq Ahmad, and Ming L. Liou

***Abstract*— Video encoding due to its high processing requirements has been traditionally done using special-purpose hardware. Software solutions have been explored but are considered to be feasible only for nonreal-time applications requiring low encoding rates. However, a software solution using a general-purpose computing system has numerous advantages: It is more available and flexible and allows experimenting with and hence improving various components of the encoder. In this paper, we present the performance of a software video encoder with MPEG-2 quality on various parallel and distributed platforms. The platforms include an Intel Paragon XP/S and an Intel iPSC/860 hypercube parallel computer as well as various networked clusters of workstations. Our encoder is portable across these platforms and uses a data-parallel approach in which parallelism is achieved by distributing each frame across the processors. The encoder is useful for both real-time and nonreal-time applications, and its performance scales according to the available number of processors. In addition, the encoder provides control over various parameters such as the size of motion search window, buffer management, and bit rate. The performance results include comparisons of execution times, speedups, and frame encoding rates on various systems.**

***Index Terms*—Motion estimation, MPEG-2, network of workstations, parallel and distributed systems, software, video coding.**

I. Introduction

Video is a fundamental component of a wide spectrum of multimedia applications. The great interest for digital—as opposed to analog—video is because it is easier to transmit, access, store, and manipulate visual information in a digital format. The key obstacle to using digitized video, however, is the enormous amount of data required to represent video in digital format. Compression of the digital video, therefore, is an inevitable solution to overcome this obstacle. Compression algorithms like MPEG-2 [3] emerged with a view to reduce the data rate to a manageable level by taking advantage of the redundancies present in both spatial and temporal domains of the digital video. Video compression requires an encoder and a decoder. An encoder compresses the data for transmission or storage, and a decoder decompresses the data to make the video in a presentable form. Since video encoding is much more complex and time-consuming than decoding, efforts are generally required to speedup the encoding process.

Video encoding can be done using special-purpose hardware, but a hardware-based approach has certain disadvantages: first, a hardware-based encoder is usually very expensive and ordinary users cannot afford it, and, thus, the number of potential users is restricted; second, a dedicated hardware is less flexible and can become obsolete; and third, a hardware-based encoder is often optimized for a particular coding algorithm, and thus does not allow exploration of other present and future algorithms.

A software solution using general-purpose computing platforms, on the other hand, is more available. The inherent modular nature of various video compression algorithms allows experimenting with and, hence, improving various parts of the encoder independently. For example, exploring new motion estimation algorithms is an active area of research, and software solutions have the flexibility to allow experimentation with such algorithms. In addition, for nonreal-time applications, a software implementation can produce better quality video compression by tuning various parameters and by allowing multiple passes for optimization. However, the very high computation requirements of video applications often overwhelm the single-processor, sequential computers [1], [11]. Therefore, exploiting the potentially enormous computing power offered by parallel and distributed technologies is a promising solution for video encoding.

A software-based encoder using multiple processors requires an efficient parallelization scheme. There have been some previous approaches ([5], [10], [11], [15]–[17]) on parallel video encoding and decoding. An MPEG-1 encoder has been reported in [17] with a special-purpose parallel hardware using digital signal processors. A parallel implementation of the H.261 video coding algorithm using a single-instruction multiple-data (SIMD) parallel machine has been reported in [10]. This system is reported to have achieved a frame rate of about 5 frames/s. Distributed load balancing schemes for a parallel video encoding system have been described in [6] involving a hybrid video-encoding algorithm recommended by CCITT SG XV for $p \times 64$ kb/s. Parallel MPEG-1 video encoding with a performance of about 4 frames/s using nine HP 9000/720 machines connected via Ethernet has been documented in [5]. Subsequently, it was modified [15] to run on the Intel Touchstone Delta and the Intel Paragon. An implementation of MPEG-2 encoding using a local area network is described in [19]; the performance of this implementation in terms of the frame encoding rate is not reported. These works, in summary, lack the prospect of real-time video encoding.

Networked clusters of workstations using communication environments such as Express, PVM, and MPI are becoming increasingly faster and cost effective. Since video compression involves processing of a large number of operations, it can be decomposed along a spatial dimension, making parallelism a logical method for handling the processing. In this paper, performance of a parallel video encoder with MPEG-2 quality on various parallel computers and networks of workstations is presented. Our objective is to make video compression algorithms architecture-independent without compromising the processing speed. The idea is to make it portable, flexible, and scalable. The portability of the code implies that the parallel software should be able to run on a vast variety of high-performance parallel supercomputers as well as general-purpose networks of workstations. Flexibility means that the performance of the software in terms of its processing speed should not depend on a particular hardware configuration, rather it should optimize itself according to the available hardware. Scalability means that by increasing the computing resources, the processing speed should increase as linearly as possible.

Manuscript received September 30, 1996; revised January 31, 1997. This paper was recommended by Guest Editors B. Sheu, C.-Y. Wu, H.-D. Lin, and M. Ghanbari.

S. M. Akramullah and M. L. Liou are with the Department of Electrical and Electronic Engineering, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong.

I. Ahmad is with the Department of Computer Science, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong.

Publisher Item Identifier S 1051-8215(97)05876-X.

The platforms used in our experimentation include an Intel Paragon/XP-S, an Intel iPSC/860 hypercube, a cluster of HP workstations, a cluster of SGI workstations, and a cluster of SUN workstations. Our encoder uses a data-parallel approach in which parallelism is achieved by distributing each frame across the processors. The encoder is useful for both real-time and nonreal-time applications, and its performance scales according to the available number of processors. The rest of the paper is arranged as follows. Section II provides a brief overview of the MPEG-2 algorithm, Section III elaborates the parallelization technique, and Section IV describes the parallel and distributed platforms. Section V includes experimental results, while the last section concludes the paper.

II. Overview of MPEG-2

As opposed to MPEG-1 which is targeted for applications requiring 1.5 Mb/s, MPEG-2 [3] is intended for a variety of applications at a rate of 2 Mb/s or above with a quality ranging from good quality National Television Standards Committee (NTSC) to high-definition television (HDTV). 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, and various qualities and services, it often needs to process large amounts of data—for example, more than 10 million pels/s according to CCIR 601 specification.

The aim of MPEG-2, such as having better picture quality while keeping the provision for random access to the coded bitstream, is a rather difficult task. Obtaining good picture quality at the bit rates 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. This leads to the definition of intra coded (I), predictive coded (P), and bidirectionally predictive coded (B) pictures [3].

The algorithm first selects an appropriate spatial resolution for the signal. It then performs motion estimation by block matching. Motion estimation refers to finding the displacement of a particular macroblock (16×16 or 16×8 pel area) of the current frame with respect to a previous or future reference frame or both of them. All searches are based on minimum absolute difference (MAD) matching criteria, i.e., on minimizing the accumulated absolute values of the pel differences for all macroblocks. To achieve temporal redundancy reduction, 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.

In order to achieve spatial redundancy reduction, the difference signal, i.e., the prediction error, is further compressed using the block transform coding technique which employs the two-dimensional (2-D) orthonormal 8×8 discrete cosine transform (DCT) 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. In MPEG-2, adaptive quantization is used at the macroblock (16×16 pel area) layer, which permits 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 nondownloadable and are therefore optimized for a limited range of compression ratios appropriate for the target applications.

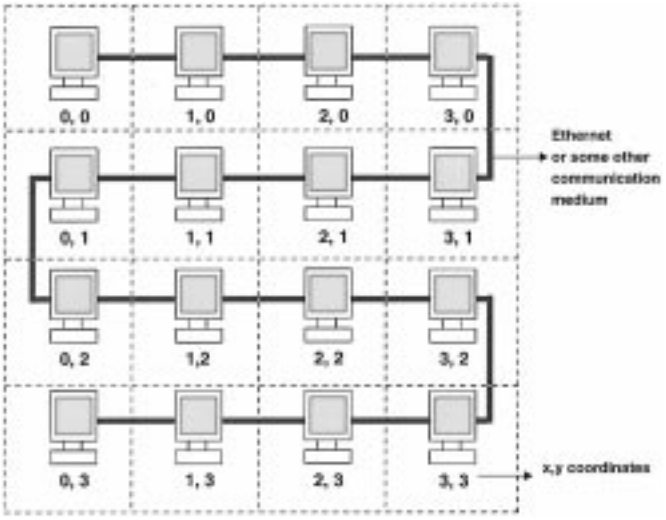

Fig. 1. The cluster of workstations connected as a two-dimensional virtual topology.

III. The Parallel Implementation

To achieve a portable and scalable parallel implementation of the MPEG-2 video encoder, we have used the *data-parallel* or *single program multiple data* (SPMD) programming paradigm. Under this paradigm, we partition[1] the frame-data into smaller pieces which are assigned to different processors. A single program is written for all processors which asynchronously execute the program on their local pieces of data. Communication of data and synchronization is done through message-passing using *Express* [13] parallel programming environment. To distribute a frame data across processors, we set up a virtual multidimensional grid of processors regardless of the underlying hardware topology. For example, a cluster of workstations connected via a bus such as Ethernet is turned into a virtual 2-D grid (see Fig. 1). The processors are assigned x–y coordinates which facilitate interprocessor communication and identification of the neighboring processors. The data is then mapped onto the virtual processor grid.

A. Data Partitioning and Processor Allocation

For efficient parallel computing, the design of distributed data structures are of particular importance. This design should be made with a view to balance the computational load on individual processors and to gain maximum parallelism of interprocessor data communications. In our approach, a frame of size $N_x \times N_y$ is partitioned into rectangular tiles of equal size $M_x \times M_y$, and each tile is assigned to one processor of the array, preserving the correspondence between nearest neighbor tiles and nearest neighbor processors. In order for the frame to fit on a rectangular processor array of size $P_x \times P_y$, we should have $P_x M_x \geq N_x$ and $P_y M_y \geq N_y$. The size of a tile $M_x \times M_y$ depends on the availability of processors. For instance, if only one processor is available, $M_x = N_x$ and $M_y = N_y$.

A frame data needs to be distributed as evenly as possible to all the processors. This can be done, for example, by just apportioning the requisite part of the frame data (one or more 16×16 macroblock) to the corresponding processors (see Fig. 2). But this method

[1] Note that in an even simpler approach [15], an entire video sequence can be partitioned into groups of pictures which can then be running multiple copies of sequential encoder on many processors. While this approach can be useful for off-line encoding, it is not suitable for real-time where it must be done on a frame-by-frame basis.

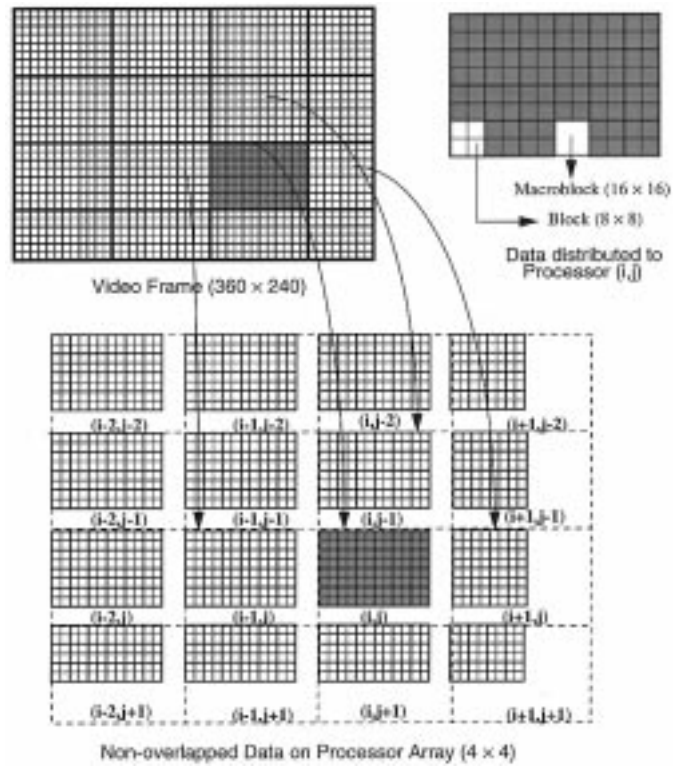

Fig. 2. Distribution of a frame on the virtual network topology.

TABLE I
PICTURE QUALITY IN TERMS OF PSNR

Sequence	Average Luminance PSNR (dB)
Football	30.3
Table Tennis	31.1
Salesman	30.3
Miss America	32.2
Swing	32.7

TABLE II
FRAME ENCODING RATE ON SGI WORKSTATIONS (FRAMES/S)

Sequence	Number of Workstations				
	1	2	4	8	16
Football	0.3065	0.5843	1.0004	1.5794	2.2038
Miss America	0.3026	0.5752	0.9931	1.5532	2.1693
Table Tennis	0.3079	0.5878	1.0057	1.5907	2.2173
Salesman	0.2872	0.5417	0.9314	1.4703	2.0850
Swing	0.3070	0.5857	1.0110	1.5949	2.2333
Average	0.3022	0.5749	0.9883	1.5577	2.1817

TABLE III
FRAME ENCODING RATE ON SUN WORKSTATIONS (FRAMES/S)

Sequence	Number of Workstations				
	1	2	4	8	16
Football	0.2152	0.4061	0.7067	1.1762	2.0162
Miss America	0.2038	0.3916	0.7068	1.1729	1.9505
Table Tennis	0.2203	0.4232	0.7708	1.2439	2.0176
Salesman	0.2067	0.3999	0.7188	1.0454	1.9287
Swing	0.2078	0.4034	0.7359	1.0949	2.0963
Average	0.2108	0.4048	0.7278	1.1467	2.0019

TABLE IV
FRAME ENCODING RATE ON HP WORKSTATIONS (FRAMES/S)

Sequence	Number of Workstations		
	1	2	4
Football	0.2813	0.5361	0.9684
Miss America	0.2728	0.5201	0.9556
Table Tennis	0.2861	0.5462	0.9718
Salesman	0.2609	0.4989	0.9213
Swing	0.2803	0.5324	0.9788
Average	0.2763	0.5267	0.9592

necessitates excessive communication among the processors as the search window (during motion estimation) moves to the boundary. On the contrary, usually each processor has enough memory to store the entire search window. Taking this into account, it is possible to eliminate excessive communication. In this case, the frame data is distributed among the processors allowing overlap such that each processor is allocated some redundant data, depending on the user-supplied size of the motion search window.

Let P and Q 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 being the number of processors in the horizontal dimension and p_v being the number of processors in the vertical dimension of the 2-D grid (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 local frame size in each processor, which is given by

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

We use this approach because it avoids interprocessor communication during motion estimation. In our implementation, the number of processors to be used is an input parameter, allowing the control of the granularity (workload assigned to each processor) of the problem. 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. Using our partitioning scheme, a calculation reveals that the maximum number of processors that can be used is 330 [1] (for a frame size of 360 × 240).

B. Implementation Features

In our parallel implementation, motion compensation is done by forming a prediction for each nonintra coded macroblock from previous or future reference frames. For the forward or backward prediction types, the final prediction is formed in-place in a single step. For the other prediction types, two steps are involved, where each step is like the forward/backward prediction; the final prediction is formed by averaging two independent predictions from the two steps.

For motion estimation, the search window size is a user-defined parameter. By using the estimated motion vectors and the reference frames, the prediction for the whole frame is constructed. Intracoded macroblocks, without having motion vectors, are treated as having a constant prediction of arbitrary value (128 in this implementation).

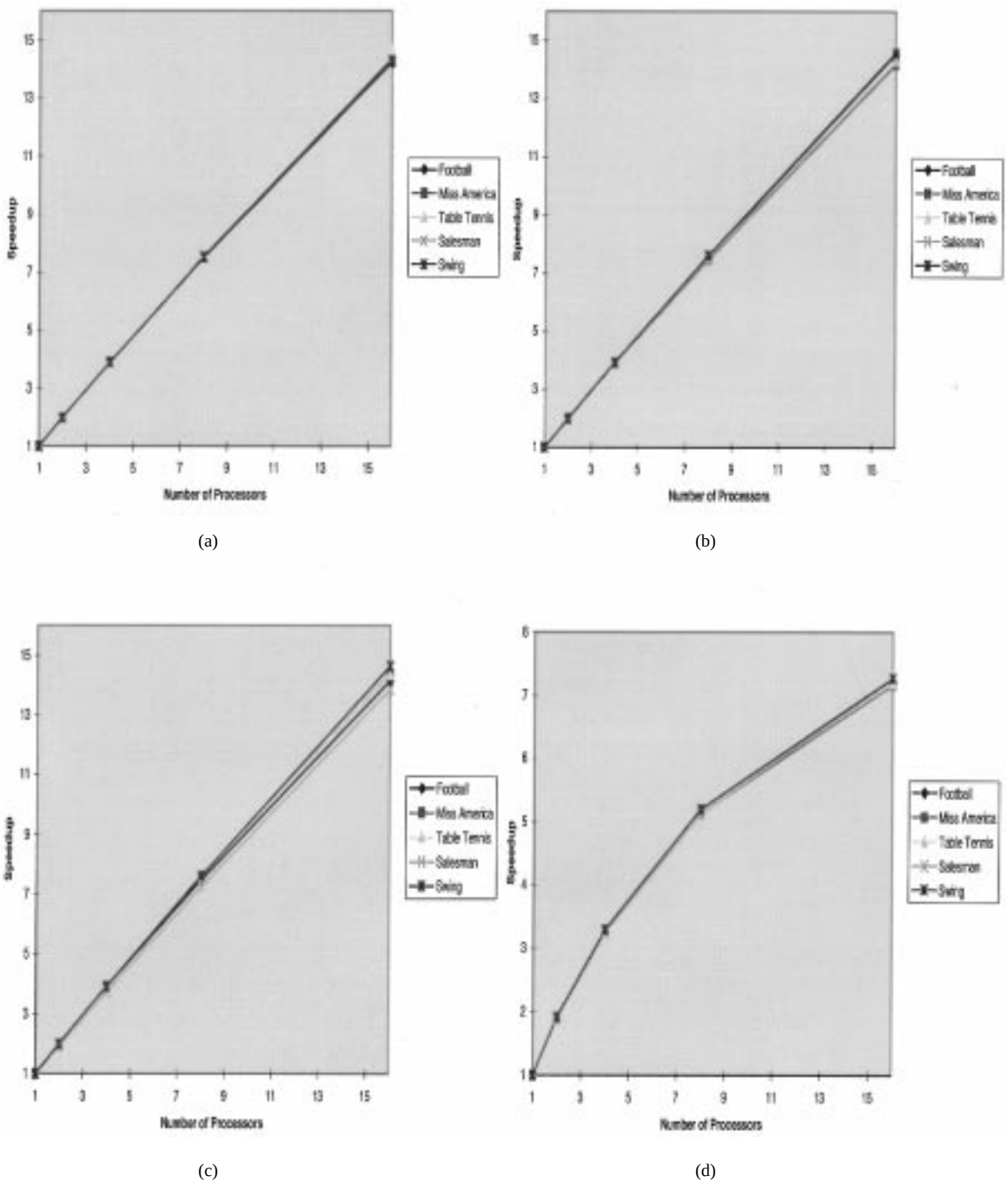

Fig. 3. Speedup of (a) motion estimation, (b) DCT, (c) quantization and VLC and (d) overall speedup on the SGI cluster.

The prediction error is transformed into frequency domain by applying DCT. For both DCT and inverse DCT (IDCT), we distribute the data such that each processor performs DCT and/or IDCT on one or more 8×8 pixel-block of data. In order to have a fast implementation and to cope with the IEEE specification of IDCT accuracy, we have used Wang's fast algorithm [18] with double precision.

Quantization and variable length coding are treated as a single module, since the rate control strategy requires highly localized feedback from the coded data buffer in order to derive the macroblock quantization scale factor. Therefore, quantization is done in parallel with the final macroblock coding stages. The strategy is similar to the one described in [4].

The allocation of target bits for the current picture being coded 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. The number of target bits for the current picture is broadcasted to all the processors. All the processors then devise their local target bits based on the received target bits for the whole picture and according to the amount of data available in that processor. The local buffer fullness in each processor is updated on a macroblock basis.

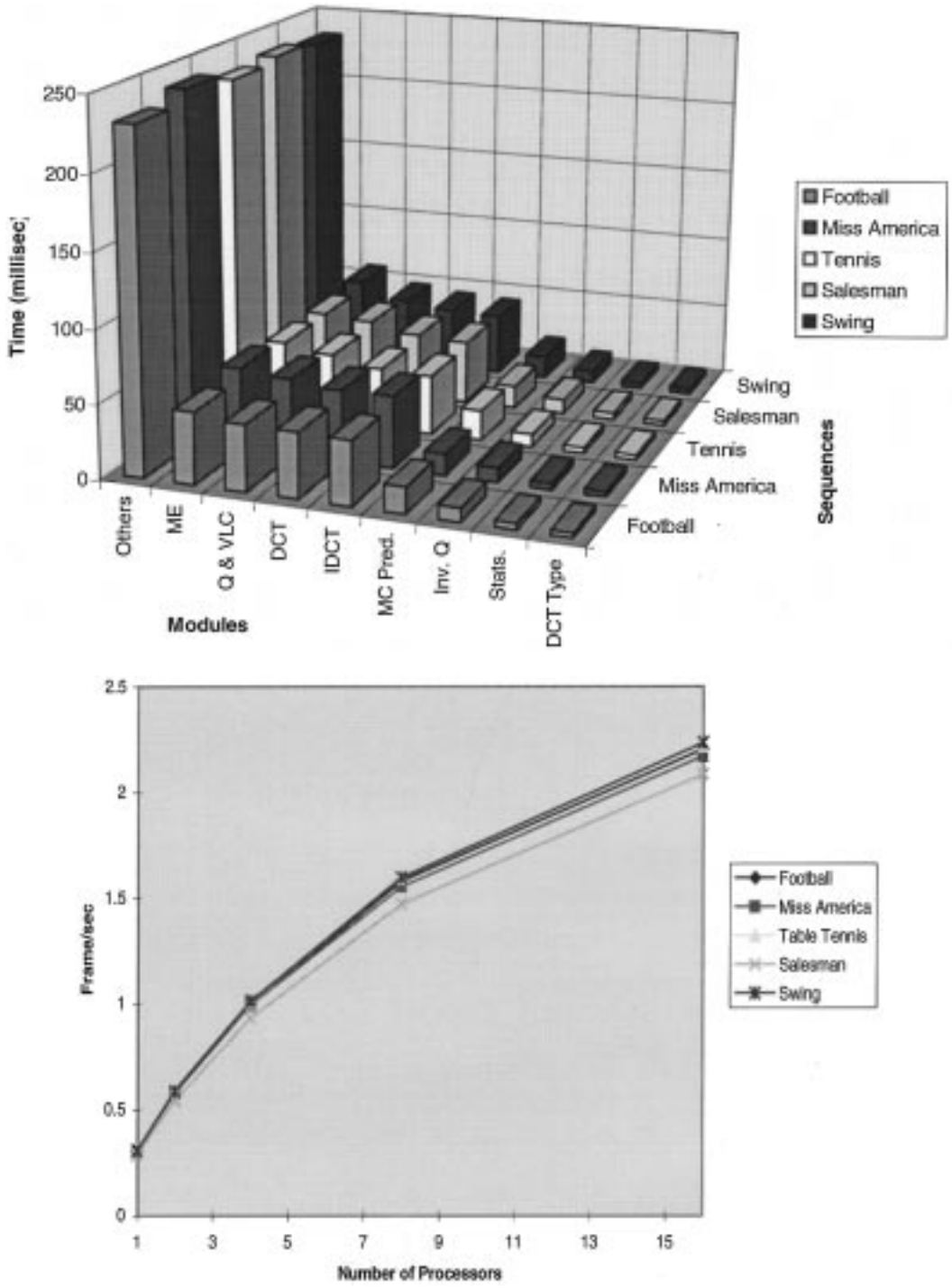

Fig. 4. Comparison of modules and frame processing speed on the SGI cluster.

The coding complexity is estimated locally as the product of the average macroblock quantization step size and the number of bits generated by each processor. The local bit allocation for the current macroblock is based on the deviance from estimated buffer fullness (i.e., the product of macroblock number and the average bits per macroblock) in a particular processor and the normalized spatial activity, similar to the sequential implementation [12]. The variance of a macroblock (calculated earlier on the four 8×8 luminance blocks, regardless of the ultimate macroblock prediction mode) has been chosen as the spatial activity measure to perform fine tuning of the quantization step size. The macroblock quantization scale is adapted according to the deviation of the locally generated bits and the estimated uniform distribution of bits in each processor, controlled by a compensation factor (difference between predicted and true buffer fullness of the current macroblock).

IV. Parallel and Distributed Platforms

In our experiments, we used five systems which are briefly described below.

The Network of SUN Workstations: The cluster of Sun workstations includes two Sparcstation 2, two Sparcstation IPX, and twelve Sparcstation 10. All 16 workstations are connected to each other via a 10-Mb/s switched Ethernet.

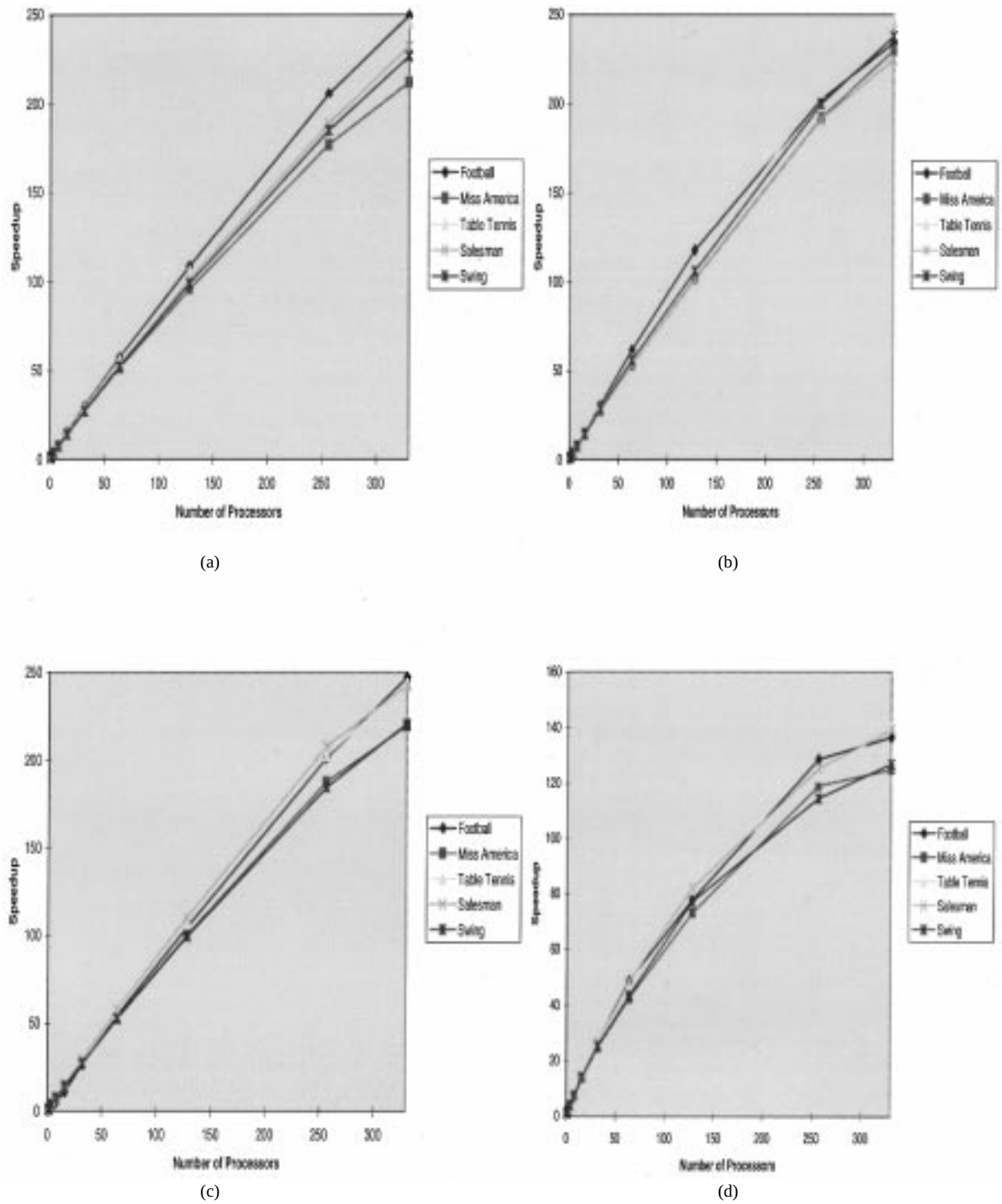

Fig. 5. Speedup of (a) motion estimation, (b) DCT, (c) quantization and VLC and (d) overall speedup on the paragon.

The Network of SGI Workstations: The network of SGI workstations consists of 16 SGI workstations of type SGI-5 connected via a 10-Mb/s switched Ethernet. They all use similar processors with homogeneous hardware capabilities.

The Network of HP Workstations: This cluster comprises four HP workstations linked with a fiber distributed data interface (FDDI) ring giving an aggregate bandwidth of 3.6 Gb/s. Each workstation is equipped with 144-Mbyte memory, a 400-Mbyte system disk, a 1-Gbyte *SCSI* disk, and a 2-Gbyte fast wide *SCSI* disk for storage of user's data. The CPU of the HP workstations are HP 9000 model 735. The HP 735 machines use 99-MHz Precision Architecture RISC (PA-RISC) 7100 microprocessor.

The Intel iPSC/860 Hypercube: The Intel iPSC/860 is a message-passing multiple instructions stream, multiple data stream (MIMD) distributed-memory parallel system consisting of compute nodes and a host computer [7]. By configuration, the compute nodes are independent processor/memory pairs with capability of running distinct programs concurrently. These nodes are connected as a *hypercube.*

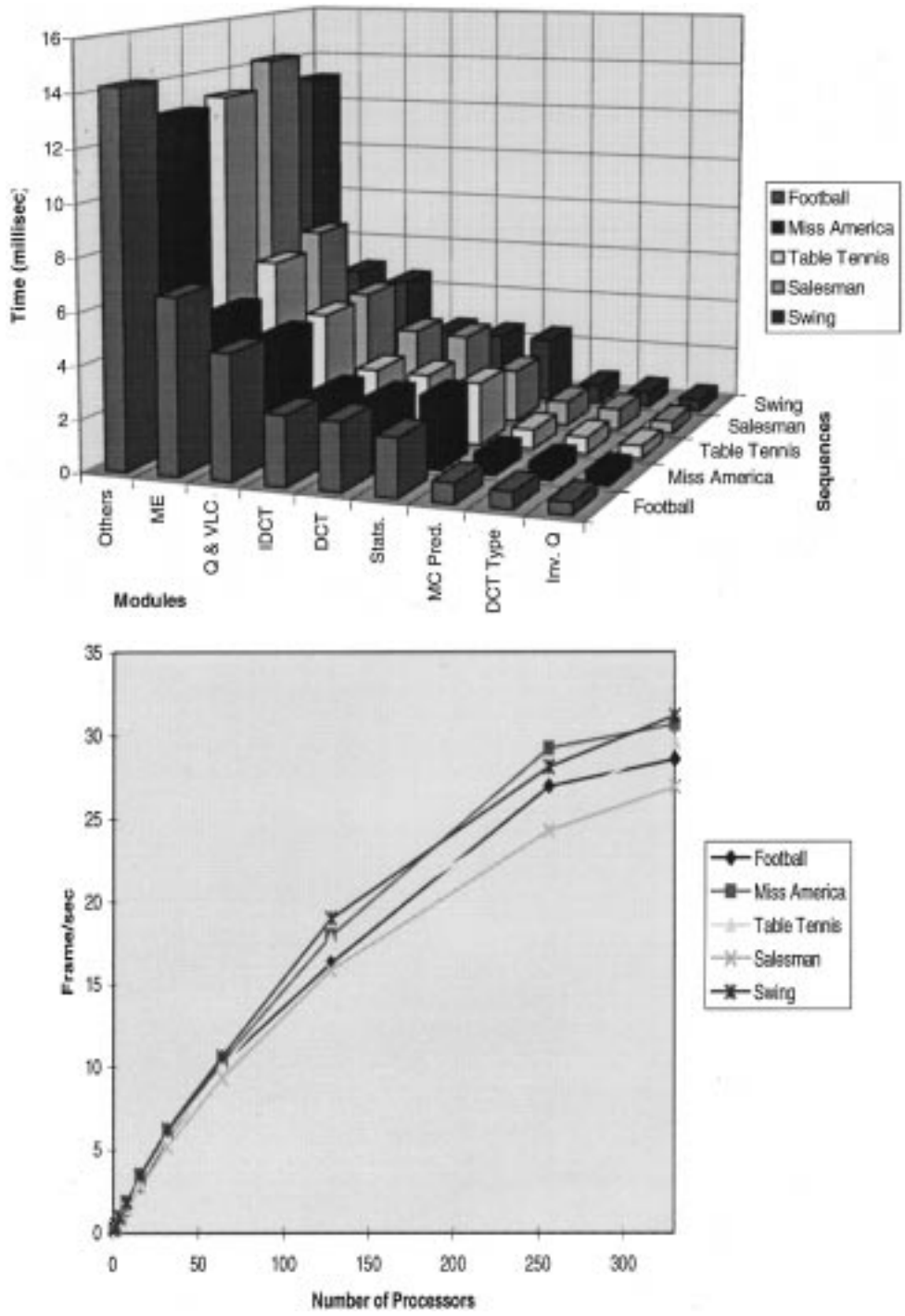

Fig. 6. Comparison of modules and frame processing speed on the paragon.

Each node consists of a 32-b i860 RISC processor with up to 16 Mbyte of local memory. The compute nodes can be thought of belonging to a node network, where each node interfaces to the node network through its direct-connect module (DCM). Essentially, communication between nearest neighbors occurs over the DCM channel using a circuit-switched routing method.

The Intel Paragon XP/S: The Intel Paragon XP/S [8] is also a distributed-memory MIMD machine and was first delivered in September 1992. Its processing nodes are arranged in a 2-D rectangular grid. The system consists of three types of nodes: compute nodes for executing parallel programs; service nodes offering capabilities of a UNIX system, including compilers and program development tools; and I/O nodes providing interface to mass storage and LAN's. All three types of nodes are implemented by the same general purpose (GP) node hardware, the i860 XP using a clock speed of 50 MHz.

Paragon's mesh routing chips (MRC's) connected by high-speed channels are the basis of the communication network connecting the nodes. There are two independent 16-b wide channels—one for each direction—between two neighboring nodes. The MRC's using wormhole routing can route messages autonomously and are independent of the attached nodes. For our experiments, we have

TABLE V
FRAME ENCODING RATE ON THE INTEL PARAGON (FRAMES/S)

Sequence	Number of Processors									
	1	2	4	8	16	32	64	128	256	330
Football	0.2094	0.3878	0.7177	1.4193	2.8528	5.6638	10.1947	16.2681	26.9324	28.5551
Miss America	0.2457	0.4826	0.9326	1.8057	3.4001	6.0669	10.3681	17.9244	29.2312	30.6466
Table Tennis	0.2109	0.4185	0.8247	1.5936	3.0408	5.7820	10.0462	17.6741	25.7865	29.5596
Salesman	0.1936	0.3842	0.7522	1.4495	2.7733	5.1698	9.3379	15.8831	24.2660	26.9034
Swing	0.2457	0.4835	0.9472	1.8271	3.4855	6.1973	10.6326	18.9573	28.1057	31.1429
Average	0.2211	0.4313	0.8349	1.6190	3.1105	5.7760	10.1159	17.3414	26.8644	29.3615

TABLE VI
FRAME ENCODING RATE ON THE INTEL IPSC/860 (FRAMES/S)

Sequence	Number of Processors						
	1	2	4	8	16	32	64
Football	0.1490	0.2882	0.5451	0.9819	1.7178	2.7381	3.9888
Miss America	0.1532	0.2970	0.5659	1.0095	1.7520	2.7601	4.0994
Table Tennis	0.1533	0.2970	0.5653	1.0169	1.7385	2.8092	4.1063
Salesman	0.1521	0.2947	0.5626	1.0076	1.7172	2.7333	4.0389
Swing	0.1588	0.3079	0.5834	1.0761	1.8278	2.9397	4.2541
Average	0.1533	0.2970	0.5645	1.0184	1.7507	2.7961	4.0975

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.

V. EXPERIMENTAL RESULTS

Experiments were performed on all five platforms described above, using various numbers of processors. We used five video sequences[2]: *Football* (360 × 240), *Table Tennis*, (360 × 240), *Salesman*, (360 × 288), *Miss America*, (352 × 288), and *Swing* (352 × 288). These sequences are representative of different kinds of motion and are very useful for testing motion estimation.

The measured time was averaged over 50 frames of a video sequence. The time to process 50 frames of a video sequence was not necessarily the same in each processor, so the average was also taken over all the processors. All of the timings were measured with microsecond precision.

We used a constant bit rate of 5 Mb/s and a *video-buffering-verifier* buffer size of 112 as input for all 50 frames, with a group of pictures (GOP) of 12 and an I-to-P frame distance of three, while the search window was ±11 pels for P-pictures and ±10 pels for B-pictures. With a view to achieve speedup in motion estimation, 2-D-logarithmic search [9] was 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 unique metric for this measure [5]. The average values of PSNR obtained for the luminance frames of different sequences using our encoder are shown in Table I.

[2] Although picture resolutions specified by CCIR601 (720 × 480 or 720 × 576) are often used for MPEG-2 implementations, our experiments involved smaller sized pictures described above. However, since our implementation is scalable, it can be performed on pictures of any resolution. In order to achieve maximum parallelism, larger pictures should be divided more (up to a single macroblock).

Note that there is no performance loss due to parallelization since, in our parallel programming model, various MPEG-2 modules in each processor use the same computational logic that is used in the sequential version; the magnitude of PSNR is dependent on the motion estimation algorithm and could be improved with an exhaustive search technique.

The frame encoding rate using various numbers of processors on the network of SGI workstations is shown in Table II.

Fig. 3 shows the speedups of various computational modules as well as the overall speedup. The linearly increasing curves for the modules show that use of more processors will result in greater speed of processing a picture. Even the curve for the overall computation is almost linearly increasing, indicating that if more processors are involved, less time will be required to encode a frame.

Fig. 4 illustrates the comparison of actual time taken to complete various MPEG-2 modules and a plot of the frame encoding rate. It shows that "the others" is the most time-consuming module. This module, in fact, consists of some housekeeping functions required to build the parallel environment, for example, decomposition of the processor array into a 2-D mesh and to start the parallel daemons. It also involves all the initialization, allocation of memory, validating input parameters according to their relations, writing header information in the bitstream, shuffling of reference frames during coding, putting the sequence end code in the bitstream, etc. Therefore, the time taken by this module depends on the parallel platform.

The plot of the encoding rate clearly shows that as the number of processors increases, the encoder is able to process more frames per second. The shapes of the curves are almost linear and it is expected that if enough number of processors are available, the encoder can perform real-time MPEG-2 video encoding. In other words, if more processors are involved and work load is divided to each processor, we can encode larger frame sizes at the same expense of time. A simple calculation can determine the number of processors needed for

real-time video encoding, and it turns out that a network of about 215 general-purpose SGI workstations will suffice to achieve real-time video encoding. However, for larger frame sizes, more workstations will be required.

The frame encoding rate on the network of SUN workstations is shown in Table III. Compared to the SGI cluster, the performance of the SUN cluster is slightly lower. A simple calculation reveals that a network of 230 general-purpose SUN workstations will be required for real-time video.

The frame encoding rate on the network of HP workstations is shown in Table IV. The performance of the HP cluster is comparable to that of the SGI cluster and better than that of the SUN cluster. With a rough estimation, we can expect a network of 124 general-purpose HP workstations to achieve real-time video encoding.

The frame encoding rate on the Intel Paragon is shown in Table V indicating a real-time encoding rate using 330 processors. Even with 256 processors, the encoding rate is close to real-time. Fig. 5 depicts the speedups of various computational modules and the overall speedup. The results indicate that the encoding rate increases almost linearly with an increase in the number of processors up to 128 processors. The gain in performance is not linear beyond that. Fig. 6 shows the relative time taken by each module and the frame encoding rate. This figure is again consistent with the results of the SGI cluster, that is, various components of MPEG-2 are parallelized efficiently but the performance bottleneck is the overhead due to parallelization.

The frame encoding rate on the Intel iPSC/860 is shown in Table VI. These results are not as encouraging as those on the Paragon but indicate that workstation clusters can outperform some of the parallel machines. The linear nature of the curves indicate that real-time processing can be achieved using about 452 processors of the hypercube.

VI. Conclusions

We have presented the implementation and performance of a parallel MPEG-2 video encoder on various parallel and distributed platforms. In our implementation, video data within each frame are distributed among the processors such that both real-time and nonreal time operations are possible. The encoder is suitable for on-line as well as off-line video encoding. Execution of the same encoder on various parallel platforms ensures portability of the program. Different combinations of processors have been used and the results show that the problem does scale as the number of processors increases. Almost linear speedup in processing is achieved on all the platforms.

The results indicate that various components of MPEG-2 can be parallelized very efficiently. Also, a comparison of the workstation clusters with parallel machines indicate that the former can be a viable choice for video encoding provided the overhead of parallelization is minimized. Since high-performance workstations are becoming increasingly faster, real-time video encoding is possible with sufficiently large number of such workstations and further optimization in the software. Our current efforts are directed in that direction

References

[1] 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, Dept. Electr. Electronic Eng., July 1995.

[2] L. Chiariglione, "The development of an integrated audiovisual coding standard: MPEG," *Proc. IEEE*, vol. 83, no. 2, pp. 151–157, Feb. 1995.

[3] Draft International Standard ISO/IEC 13818, "Generic coding of moving pictures and associated audio," Seoul, Nov. 1993.

[4] S. Eckart and C. Fogg, "ISO/IEC MPEG-2 software video codec," in *Proc. SPIE*, Feb. 1995, vol. 2419, pp. 100–109.

[5] K. L. Gong and L. A. Rowe, "Parallel MPEG-1 video encoding," presented at 1994 Picture Coding Symp., Sacramento, CA, Sept. 1994.

[6] Z. Huang, Y. Takeuchi, and H. Kunieda, "Distributed load balancing schemes for parallel video encoding system," *IEICE Trans. Fundamental Electron. Commun. Comput. Sci.*, vol. E77-A, no. 5, pp. 923–930, May 1994.

[7] Intel Supercomputer Systems Division, Intel Corporation, *iPSC/860 System User's Guide*, Mar. 1992.

[8] Intel Supercomputer Systems Division, Intel Corporation, Intel Paragon XP/S technical summary, Jan. 1994.

[9] J. R. Jain and A. K. Jain, "Displacement measurement and its application in interframe image coding," *IEEE Trans. Commun.*, vol. 29, pp. 1799–1808, Dec. 1981.

[10] A. C. P. Loui, A. T. Ogielski, and M. L. Liou, "A parallel implementation of the H.261 video coding algorithm," in *Proc. IEEE Workshop on VSPC*, Raleigh, NC, Sept. 1992, pp. 80–85.

[11] P. Moulin, A. T. Ogielski, G. Lilienfeld, and J. W. Woods, "Video signal processing and coding on data-parallel computers," *Digital Signal Processing*, vol. 5, no. 2, pp. 118–129, Apr. 1995.

[12] MPEG Software Simulation Group, *MPEG-2 Video Encoder, Version 1.1a*, July 1994.

[13] Parasoft Corporation, *Express System User's Guide*. Pasadena, CA, 1992.

[14] P. Pirsch and H. Jeschke, "A MIMD multiprocessor system for real-time image processing," in *Proc. SPIE*, vol. 1452, pp. 544–555.

[15] K. Shen, L. A. Rowe, and E. J. Delp, "A parallel implementation of an MPEG1 encoder: Faster than real-time," in *Proc. SPIE*, Digital Video Compression: Algorithms and Techniques, San Jose, CA, Feb. 1995, vol. 2419.

[16] F. Sijstermans and J. van der Meer, "CD-I full-motion video encoding on a parallel computer," *Commun. ACM*, vol. 34, no. 4, pp. 81–91, Apr. 1991.

[17] H. H. Taylor *et al.*, "An MPEG encoder implementation on the Princeton Engine video supercomputer," in *Data Compression Conference 1993.* Los Alamitos, CA: IEEE Computer Society Press, 1993, pp. 420–429.

[18] Z. Wang, "Fast algorithms for the discrete W transform and for the discrete Fourier transform," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-32, no. 4, pp. 803–816, Aug. 1984.

[19] Y. Yu and D. Anastassiou, "Software implementation of MPEG-II video encoding using socket programming in LAN," in *Proc. SPIE*, Feb. 1994, vol. 2187, pp. 229–240.