A Software-Based MPEG-4 Video Encoder Using Parallel Processing

Yong He, *Student Member, IEEE,* Ishfaq Ahmad, *Member, IEEE,* and Ming L. Liou, *Fellow, IEEE*

***Abstract*—In this paper, we describe a software-based MPEG-4 video encoder which is implemented using parallel processing on a cluster of workstations collectively working as a virtual machine. The contributions of our work are as follows. First, a hierarchical Petri-nets-based modeling methodology is proposed to capture the spatiotemporal relationships among multiple objects at different levels of an MPEG-4 video sequence. Second, a scheduling algorithm is proposed to assign video objects to workstations for encoding in parallel. The algorithm determines the execution order of video objects, ensures that the synchronization requirements among them are enforced and that presentation deadlines are met. Third, a dynamic partitioning scheme is proposed which divides an object among multiple workstations to extract additional parallelism. The scheme achieves load balancing among the workstations with a low overhead. The striking feature of our encoder is that it adjusts the allocation and partitioning of objects automatically according to the dynamic variations in the video object behavior. We have made various additional software optimizations to further speed up the computation. The performance of the encoder can scale according to the number of workstations used. With 20 workstations, the encoder yields an encoding rate higher than real time, allowing the encoding of multiple sequences simultaneously.**

***Index Terms*—Data partitioning, dynamic scheduling, load balancing, MPEG-4, parallel and distributed processing, Petri nets, video encoder.**

I. Introduction

THE current and emerging multimedia services demand many more functionalities than those offered by the traditional standards. For example, mobile communication requires very low bit-rate video coding and error resilience across various networks, virtual reality and animation require integration of natural and synthetic hybrid object coding, and interactive digital video requires a high degree of object based interactivity. Instead of traditional frame-based interaction such as fast forward, fast backward, etc., new ways of interactivity are needed to efficiently realize such applications. The new standard, MPEG-4, which is currently being developed by MPEG, will enable the integration of the production, distribution, and content access paradigms in a multimedia environment [1]. With a flexible toolbox approach, MPEG-4 is capable of supporting diverse new functionalities, and hence will cover a broad range of present and future multimedia applications.

MPEG-4, due to its content-based representation nature and flexible configuration structure, is considerably more complex than previous standards. Any MPEG-4 hardware implementation is likely to be very much application specific. Therefore, software-based implementation seems to be a natural and viable option. In addition, a software-based approach allows flexibility, portability, scalability, and permits the inclusion of new tools, which are extremely desirable features for MPEG-4-based interactive multimedia systems. The main obstacle in such an approach is that it requires a large amount of computing power to support real-time encoding and decoding operations. However, the latest developments in parallel and distributed systems promise a higher degree of performance at an affordable cost (such as a network of workstations or PC's), provided the parallelism from the application at hand is effectively extracted.

A parallel implementation of the MPEG-4 encoder is a nontrivial task, and cannot be accomplished using a straightforward multitasking or data-partitioning strategy. This is because objects in MPEG-4 video add or drop from a video scene, with their sizes varying from time to time. In addition, various objects need to be tightly synchronized. Finally, depending upon the application requirements, MPEG-4 allows us to adopt different encoding efficiencies and levels of scalability on various objects. Orchestrating various tasks of the encoder and distributing and dividing objects into pieces for concurrent execution pose some research challenges. Thus, parallelization of the MPEG-4 encoder requires highly efficient scheduling and a load-balancing scheme. An effective implementation of the encoder also needs modeling tools that can capture the spatiotemporal relationships between different MPEG-4 objects.

We are currently building an MPEG-4-based interactive multimedia environment for supporting applications in the areas of CAD, teaching, and multimedia authoring. As a part of this system, we have implemented an MPEG-4 encoder with a software-based approach using parallel processing on a cluster of workstations. The main contributions of our work include the following.

- A Petri-net-based modeling scheme for capturing the spatiotemporal relationships between MPEG-4 video components at various levels (video session, object, or video object plane level).

Manuscript received October 31, 1997; revised May , 1998. This work was supported by the Hong Kong Telecom Institute of Information Technology. This paper was recommended by Associate Editor M.-T. Sun.

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

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

Publisher Item Identifier S 1051-8215(98)08385-2.

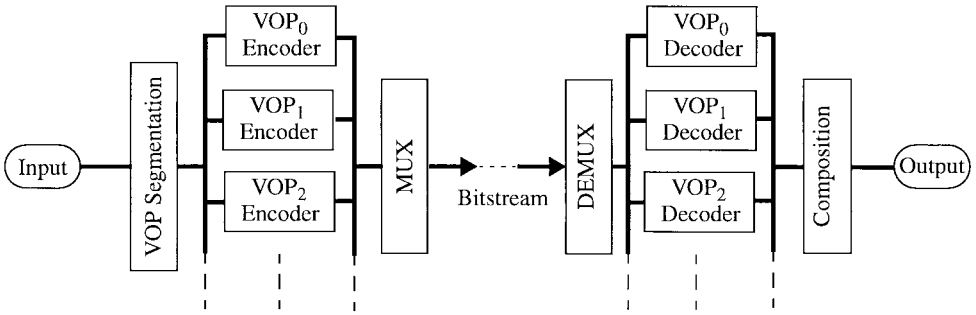


Fig. 1. MPEG-4 video codec (encoder and decoder) structure.

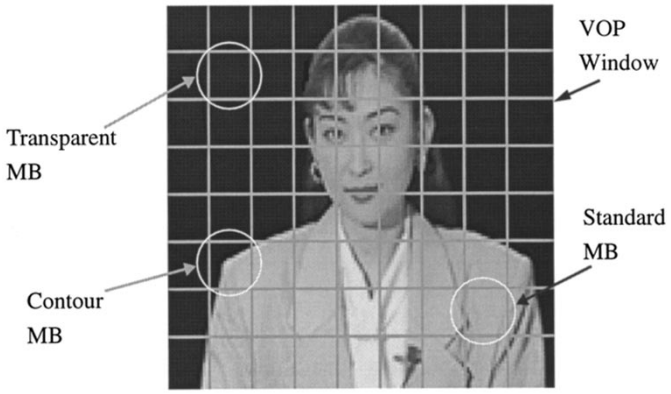
- Efficient parallel processing of the encoder through an effective scheduling algorithm. The algorithm uses the information generated by the model to allocate objects to workstations for parallel encoding, and ensures that the synchronization requirements among various objects are observed, presentation deadlines are met with a guarantee of quality of service, and that maximum speed up is obtained in terms of compression time. Allocating objects to workstations for concurrent processing is equivalent to exploiting *control parallelism*.
- A dynamic and adaptive data-partitioning scheme that maximizes the parallelism by further dividing an object among multiple workstations is proposed. Since the size of a video object may change from time to time, the partitioning strategy ensures load balancing among the divided parts. Division of a video object among multiple workstations is equivalent to exploiting *data parallelism*.

Our encoder encodes various input video objects by adjusting the allocation and partitioning of objects automatically regardless of the dynamic variation of the video object behavior. Various levels of software optimization have been used to speed up the computation. The performance of the encoder can scale according to the number of workstations used. With 20 workstations, the encoder yields an encoding rate higher than real time on some sequences. This allows us to encode multiple sequences at the same time.

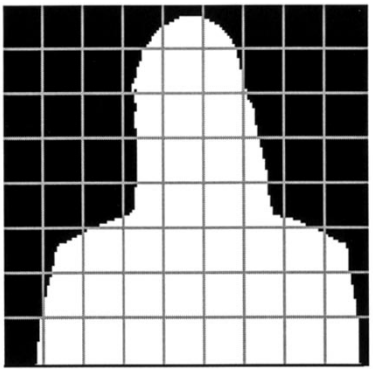
The rest of this paper is arranged in the following manner: Section II gives a brief overview of MPEG-4 video verification model. Section III describes the proposed implementation approach in detail, including: 1) a Petri nets modeling methodology introduced to model and represent the timing constraints of the video session, 2) an effective scheduling algorithm which schedules various subtasks of the encoder, and 3) a dynamic data-partitioning scheme used for further speed-up gain. Section IV provides the experimental results, and the last section presents the conclusion.

II. Overview of MPEG-4 Video

MPEG-4 is scheduled to become an international standard in November 1998. As one of the major parts of MPEG-4, MPEG-4 video is an object-based hybrid natural and synthetic coding standard which specifies the technologies enabling the functionalities such as content-based interactivity, efficient compression, error resilience, and object scalability [2]. Fig. 1 is the overall structure of MPEG-4 video codec (encoder and decoder) which is based on the concept of video object planes (VOP's) defined as the instances of video objects.



(a)



(b)

Fig. 2. Representation of the VOP (person Akiyo). (a) Image of original "Akiyo" VOP. (b) Binary alpha phase of "Akiyo" VOP.

The video encoder is composed of a number of VOP encoders as is the decoder. The same coding scheme is applied to each video object separately, and the reconstructed video objects are composited together and presented to the user. The user interaction with the objects such as scaling, dragging, and linking can be handled either in the encoder or in the decoder.

In order to describe the arbitrarily shaped VOP's, MPEG-4 defines a VOP by means of a bounding rectangle called a "VOP window." The window surrounds the VOP with the minimum number of macroblocks, as depicted in Fig. 2(a). There are three kinds of macroblock (MB) within a VOP window:

the transparent MB, the contour MB, and the standard MB. The contour and standard MB include the pixels belonging to the object, and the transparent MB lies completely outside the object area.

Each VOP encoder consists of three main parts: shape coding, motion estimation/compensation, and texture coding. Shape information of VOP is referred to as alpha plane in MPEG-4. As Fig. 2(b) shows, the alpha plane has the same format as the luminance file, and its data indicate the characteristics of the relevant pixels (inside or outside the object). The shape coder performs the compression on the alpha plane. Because the transparent MB has no object pixels inside, it will not be processed for the motion and/or texture coding.

Motion estimation and compensation (ME/MC) are used to reduce temporal redundancies. A padding technique is applied on the reference VOP which allows polygon matching instead of block matching for rectangular image. SAD (sum of absolute difference) is used as the error measure, and is calculated only on the pixels inside the object. SAD is given by

$$\begin{aligned}\mathrm{SAD}_N(x,y) = &\sum_{i=1,j=1}^{N,N} |\mathrm{original} - \mathrm{previous}| \\ &* (!(\mathrm{alpha}_{\mathrm{original}} \equiv 0)), \\ x,y = [-64,63], &N = 16 \quad \text{or} \quad 8.\end{aligned}$$

In addition to the basic motion technique, unrestricted ME/MC, the advanced prediction mode, and bidirectional ME/MC (especially for the B frame) are supported by the MPEG-4 video to obtain a significant quality improvement with a little increase in complexity.

The intra and residual data after motion compensation of VOP's are coded by texture-coding algorithms including DCT or shape-adaptive DCT (SA-DCT), MPEG or H.263 quantization, intra dc and ac prediction, and VLC to achieve further compression. For contour MB's, a low-pass extrapolation padding technique is employed before performing DCT.

MPEG-4 also supports the scalable coding of video objects in both spatial and temporal domains, and provides error resilience across various media. In addition to the above basic technologies used in the encoder structure, the toolbox approach of MPEG-4 video makes it possible to achieve more improvement for some special cases by using dedicated tools. Further details on MPEG-4 video encoder can be found in [3].

III. Proposed Implementation Approach

Most multimedia applications have real-time requirements which demand the codec to be highly efficient. In order to deal with arbitrarily shaped objects, more sophisticated techniques are needed to achieve an efficient compression. This, however, can introduce extra complexity in the encoder, which in turn requires additional computational power. Since the encoder of MPEG-4 video is much more complex and time consuming in computing than the decoder, it is more challenging to speed up the computation in the encoder.

As mentioned earlier, the hardware-based MPEG-4 encoder is likely to be very much application specific. The flexible and extensible nature of MPEG-4 requires a highly flexible and programmable encoder which is more feasible using a software-based approach. But the computational requirement of a software-based encoder is simply too enormous to be handled by a single processor PC or a workstation. It is, therefore, natural to exploit the high computational power offered by a high-performance parallel or distributed system. The architecture of MPEG-4 encoder as shown in Fig. 1 also happens to be very suitable for distributed computing. Each input VOP is encoded separately, and efficient performance can be achieved by decomposing the whole encoder into separate tasks with individual VOP encoders and running these tasks simultaneously.

In a simpler approach, one could use a single workstation to encode one VOP, but this scheme does not fully exploit the computational power of the system because it is not scalable and the degree of parallelism is rather limited. A more effective approach is to form groups of workstations, with each group working on a single VOP while additional parallelism is exploited by partitioning a VOP among the workstations within the group. This scheme, however, requires a careful division of video objects as they are interrelated. Furthermore, the sizes of VOP's change with time, implying that distribution and partitioning of VOP's will need to be adjusted accordingly. Since this must be done in real time, the cost of scheduling and distribution must be kept low to ensure that the benefits gained from an efficient parallelization are not outweighed by a lengthy scheduling time.

In our scheme, we divide a given number of workstations to groups, and assign the encoding task of one VOP to one group. However, when the VOP's are distributed to different groups of workstations, the spatiotemporal relationships between various VOP's must be preserved. Such relationships can be kept to an extremely detailed level by using a Petri nets model which is described below. A scheduling algorithm is proposed to allocate the workstations proportionally, and to decide their execution sequence in accordance with the priority of each VOP so that the timing constraint can be satisfied.

The data of a VOP are divided among the workstations within a group, allowing further gain in computing speed. For distributing the data of a VOP, various simple partitioning schemes are possible. We propose a shape-adaptive data-partitioning scheme that ensures load balancing among the divided pieces and incurs a low overhead. The details of the modeling methodology, scheduling algorithm, and data-partitioning scheme are described next.

A. Modeling MPEG-4 Video Sequence with Petri Nets

In MPEG-4 video encoder, one of the most important issues to consider is the synchronization of various video objects. Each object may have certain presentation timing constraints which, in turn, may be dependent on the other objects. The playout time requirements and associated synchronization constraints among multiple video objects must be satisfied in real time to guarantee that a smooth flow of video sequence is presented to the user.

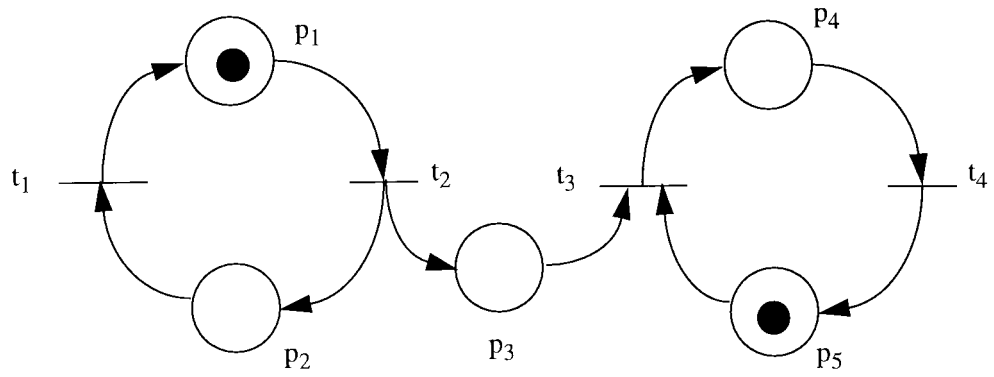


Fig. 3. Graphical representation of a Petri net.

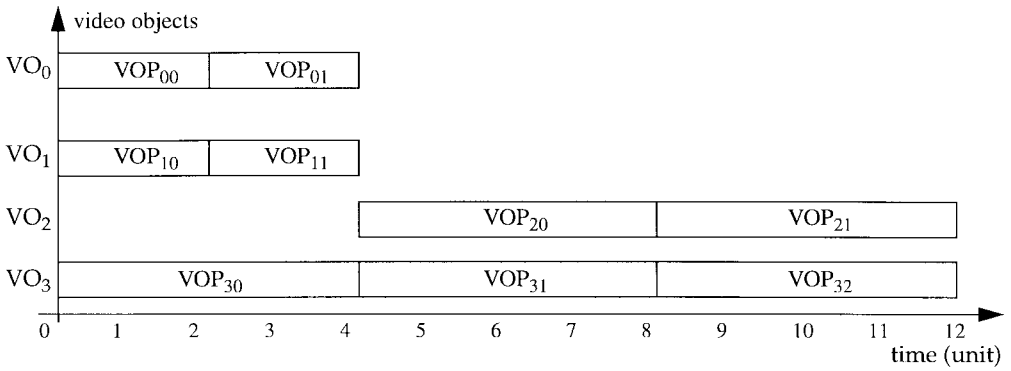


Fig. 4. Playout time chart for video session.

To identify the timing constraints among multiple objects, a synchronization reference model is required to describe temporal relationships for determining an appropriate scheduling scheme. Several modeling tools have been proposed for specifying the temporal behavior of various multimedia systems [4]. We choose Petri nets as the modeling tool since it is a simple but effective tool for describing and studying systems with concurrent, distributed, and parallel characteristics [5]. A number of variations of the Petri nets model, such as OCPN [6], XOCPN [7], and TSPN [8], have been widely used in multimedia communication applications due to its intuitive graphical representation and the simplicity of the modeling concept.

Fig. 3 depicts a graphical representation of a Petri net. The circles and bars represent the places and transitions, respectively, and the arcs indicate both input and output flow directions. A Petri net is executed by the firing rules that transmit the marks or tokens from one place to another, and such firing is enabled only when each input place has a token inside. Thus, by using firing transition and a token distribution state, Petri nets can describe the information flow or system activities in a straightforward way.

Because a Petri nets modeling structure may become complex when it is used to model a complicated real-world system, a hierarchical Petri net can be used to refine the system behavior in a step-by-step fashion. As for the MPEG-4 video session, due to its object-based nature, the number of objects presented within a scene may vary from time to time since an object can be added in or dropped out from the scene randomly. Moreover, while some of the objects may be tightly time dependent with each other, and thus must be synchronized accordingly, others may not be required to be stringently synchronized. To represent such a complex session, we can utilize hierarchical Petri nets with a similar structure as the syntax definition of MPEG-4 video which consists of video session (VS) level, video object (VO) level, video object layer (VOL) level, and video object plane (VOP) level (for the sake of simplicity in our implementation, we consider only VS, VO, and VOP levels.)

By using a hierarchical model we can achieve coarse or fine-grained synchronization by applying scheduling schemes on different levels. Fig. 4 shows the playout time chart of a general MPEG-4 video sequence. The sequence has four video objects (VO's); VO_0, VO_1, VO_3 start at time unit 0, and VO_2 starts at time unit 4. VO_0 and VO_1 are synchronized with each other and both end at time unit 4, while VO_2 and VO_3 are also synchronized and end at time unit 12. The duration of a frame for both VO_0 and VO_1 is two time units, and four for VO_2 and VO_3.

Fig. 5 represents the hierarchical Petri nets model for the above case. In the hierarchical Petri nets model, we define the place as *object intermedia unit* (OIU) and the transition as *timing constraint point* (TCP). At the VS level, an OIU represents the whole video session with just two TCP's (the session start and end point). At the VO level, each OIU represents one object within the session; here, the TCP's indicate the temporal relationship and timing constraints among various objects. At the VOP level, each OIU represents one frame of

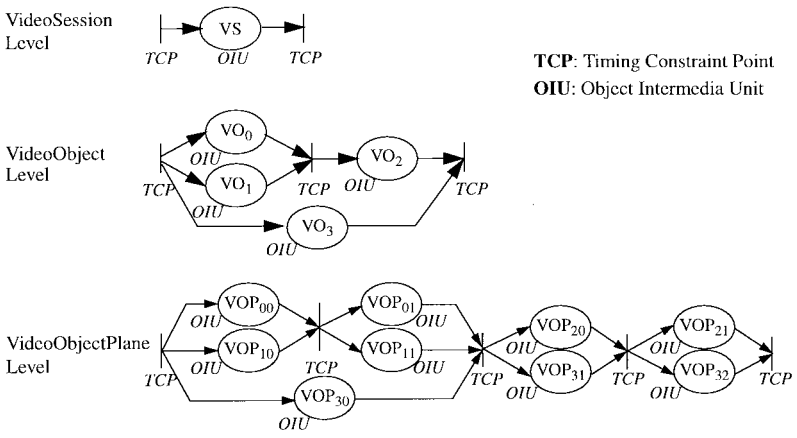


Fig. 5. Hierarchical Petri nets model of video presentation.

the object, whereas the TCP's indicate the intra and/or inter VOP's synchronization on the frame level.

Generally, all video objects play out simultaneously in the natural video session, which results in the same frame rate and presentation deadlines. For some synthetic sequences, different video objects may be introduced or halted randomly by the operations such as user interaction and content-based retrieval, and therefore may have different timing characteristics. In order to build a Petri nets model for such a dynamic behavior, we have to obtain all of the necessary information such as processing times, frame rate, object dependencies, and synchronizations beforehand. It is possible for nonreal-time applications to generate the complete Petri nets model for static scheduling with the temporal information of all video objects available in advance. For real-time applications, however, such knowledge can only be obtained at run time, and the model is generated partially along the playout sequences which, in turn, determines a dynamic scheduling scheme. For example, in the above case, we can obtain the playout deadlines and frame rates of VO_0, VO_1, and VO_3 after a short time of observation at the beginning of the session. The model can then be constructed, and it remains the same if the status of all VO's is stable. Until a new object (VO_2) is added or some existing objects (VO_0 and VO_1) are deleted at a certain time (time unit 4), the model construction can then be changed according to the new knowledge obtained after another short observation time.

B. Scheduling Objects to Workstations

We use a scheduling algorithm to allocate objects in a video session to workstations. The objective of a scheduling algorithm in a parallel processing environment is to minimize the overall execution time of a concurrent program by properly allocating the tasks to the processors and sequencing their executions [9]. A scheduling algorithm can be characterized as being either *static* or *dynamic*. A static scheduling algorithm determines the schedule with the complete knowledge of all of the tasks before the program execution. In contrast, a dynamic scheduling algorithm deals with task assignment at run time because the information about the tasks is not available in advance. Static scheduling incurs little run-time cost, but cannot adapt to the indeterministic behavior of the system. On the other hand, although dynamic scheduling is more flexible as it can be adjusted to system changes, it incurs a high run-time cost. In an MPEG-4 video session, even though a static scheduling scheme is feasible for some nonreal-time applications, it is not suitable for most real-time applications because of the unpredictable characteristics of the VOP's.

In our implementation, we have designed a hybrid static and dynamic scheduling scheme. Using the Petri nets model, the information about the video objects can be acquired after observing the objects for a short time at the beginning of each presentation period. The length of the presentation period depends on the availability of objects. During that period, the temporal characteristics of the video objects, such as frame rates and synchronizations, are assumed to be relatively stable. Therefore, we can perform a static scheduling scheme on each period with the knowledge obtained at the beginning of the period, and reschedule the new period with the updated parameters. Such a scheduling scheme combines the advantages of static and dynamic strategies, and, with a little overhead, adapts to the variation of both deterministic and indeterministic video objects. Fig. 6 depicts a scheduling scenario at VO level for the example shown in Fig. 4. The scheduling algorithm is invoked whenever a new presentation period begins. The scheduling period is bounded by the successive object scheduling instants (OSI's), and the complexity of the scheduling depends on the number of OSI's during the whole video session.

A number of scheduling algorithms have been developed for distributed and parallel systems [10]. The proposed algorithm is a variant of the *earliest deadline first* (EDF) algorithm which has been widely employed in many applications [11]. The basic principle of this algorithm is that the tasks with earlier deadlines are assigned higher priorities and are executed before tasks with lower priorities. In our implementation, VOP's with the earlier playout deadlines or synchronization constraints are encoded and delivered first. Fig. 7 shows the Petri nets model of the scheduled VOP execution order at VOP level for the case shown in Fig. 4. VOP_{00}, VOP_{10}, and VOP_{30} have a playout time of unit 0 which is earlier than those of other VOP's; thus, they are processed first. Then VOP_{01} and

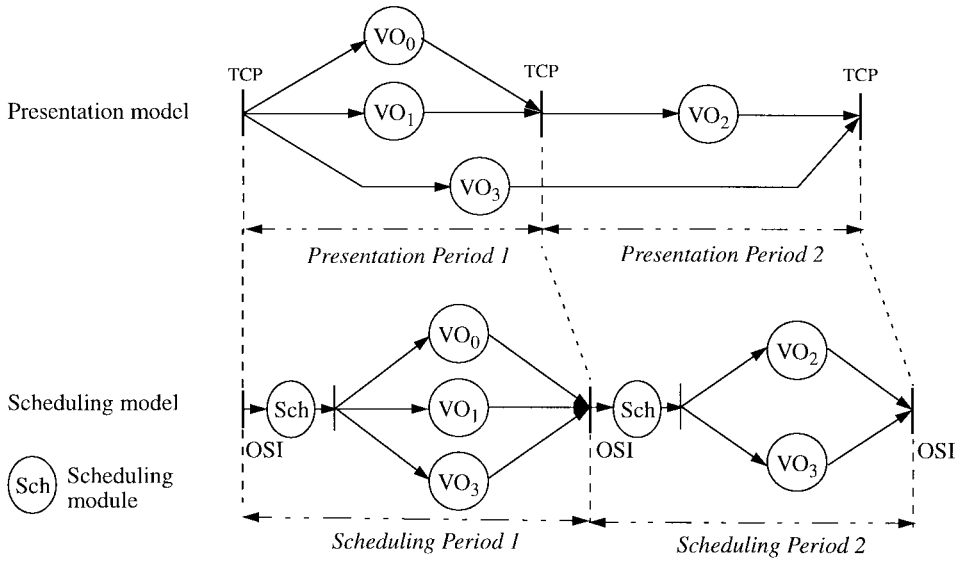


Fig. 6. Petri nets model for dynamic scheduling.

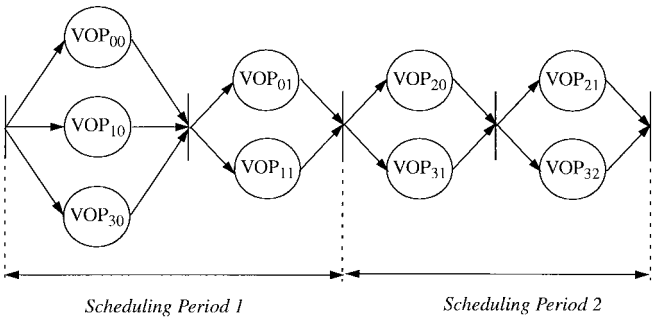


Fig. 7. Petri nets model of EDF scheduling.

VOP_{11}, whose playout deadline is time unit 2, are processed next, and so on. For the tasks with the same TCP such as VOP_{01} and VOP_{11}, since a video object with a larger size usually requires more computing power, and vice versa, we allocate the available processors to each VOP proportionally according to the VOP size ratio between the VOP's.

C. Dynamic Shape-Adaptive Data Partitioning

Parallel programming paradigms can be classified into various models such as object-oriented model, control-parallel model, and data-parallel model. The data parallel paradigm emphasizes exploiting parallelism in large data sets. The main idea of data partitioning in video encoding is to decompose the whole frame data into a number of data blocks, and map these blocks onto the corresponding processors. Since the processors perform the computation on their own data simultaneously, a high speed up can be achieved.

In a data parallel program, the issue of load balancing should be carefully addressed. Load balancing means equalization of the processors' workloads to minimize their idle times [12]. In MPEG-4 video, the size and location of each object may vary with time, and such behavior cannot be predicted beforehand. Therefore, no matter how initial tasks are assigned, the workloads of the processors will become unbalanced later on. Some processors will become highly loaded, while others are idle or lightly loaded. Furthermore, some computation-intensive algorithms of the encoder are data dependent, and their execution times are different for various data regions. For example, as depicted in Table I, some operations are performed on all macroblocks, while others just act on contour and/or standard MB's. Thus, the problem of load balancing becomes nontrivial.

TABLE I
MPEG-4 VIDEO TOOLS DEPLOYMENT

Tools	Transparent MB	Contour MB	Standard MB
Motion Estimation	N	Y	Y
CAE-Shape coding	N	Y	N
Padding (motion)	N	Y	Y
IDCT/DCT	N	Y	Y

We have developed a dynamic shape-adaptive data partition method to guarantee the workload balancing during the whole video session with low run-time overhead and fine granularity. This method can be explained by considering that the entire MPEG-4 video session can be characterized by the number of time intervals. The time interval boundary depends on the variation of the VOP window size. A new time interval begins whenever a VOP window size changes. For example, Fig. 8 shows the intervals of the test sequence "Weather" (person woman) with the number of frames ranging from 150 to 300. Since the knowledge of video objects, including the object size and the contour and standard macroblock distribution, can be obtained at the beginning of the interval. We can perform the partitioning scheme (as described below) within each time interval. During that interval, we can assume that the spatial computation distribution is relatively stable, and that there is no need to change the partition. Therefore, the proposed method can handle the object variations with a small run-time overhead. Since most of the algorithms are macroblock based,

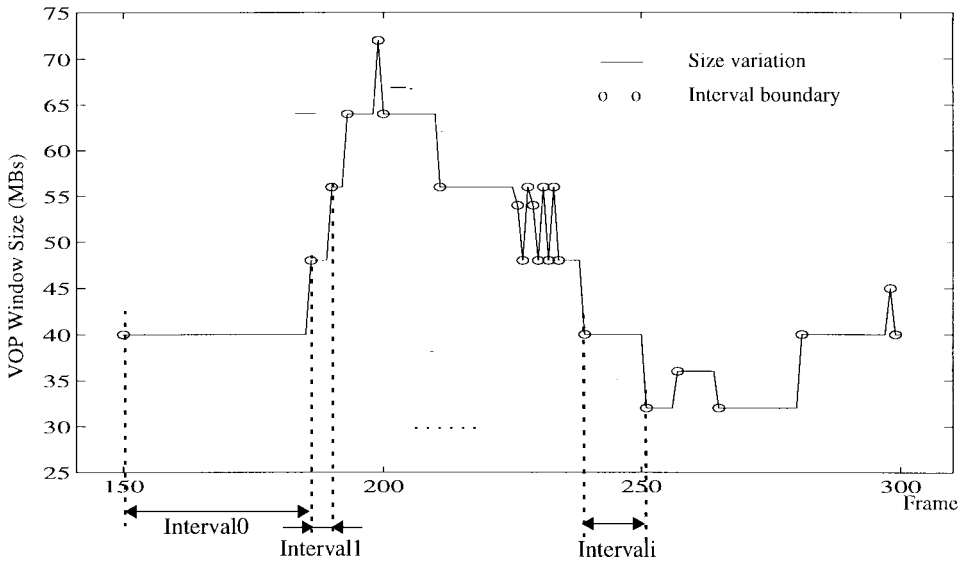


Fig. 8. Time interval example of the "Weather" sequence (person woman).

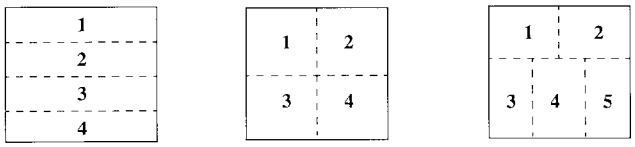


Fig. 9. Some simple partitioning methods. (a) Strip-wise decomposition. (b) Blockwise decomposition. (c) Recursive bisection.

we employ a macroblock-based data partitioning to map an integer number of macroblocks to each processor. This allows each processor to execute the compression algorithm on its own data.

In its simple form, a data-partitioning method may restrict the subregion to a rectangular shape to avoid the use of complex data structures. Fig. 9 shows some simple partitioning methods. A stripwise partition divides the whole VOP window horizontally or vertically into n subregions for n processors. It is easy to determine the area of subregions for corresponding processors, while the number of boundary pixels is high. A blockwise partition divides the VOP window evenly along both the horizontal and vertical dimensions. In this case, the number of boundary pixels of the subregion is minimal, but the number of processors to be used is restricted. The recursive bisection method [13] divides the whole VOP window recursively in a binary fashion. Although the computational load can be optimally distributed, it is relatively expensive to execute the recursive operations during the decomposition.

For MPEG-4, when an object is large enough and almost fills the entire VOP window, rectangular region partitioning methods may achieve good load balancing because the contour and standard MB's are likely to be distributed uniformly among multiprocessors. In general, some subregions of the window are full of transparent MB's, while others may be full of contour and/or standard MB's. Therefore, no partitioning method can equally distribute rectangular subregions in a straightforward way. In addition, because the object size may be too irregular, it may not be possible to employ the stripwise or blockwise partition. Fig. 10 shows a partitioning example of both stripwise and blockwise decomposition on the first frame of the test sequence "*Children*" (QCIF). With nine processors available, it is apparent that some processors are assigned almost all transparent macroblocks (which require low com-

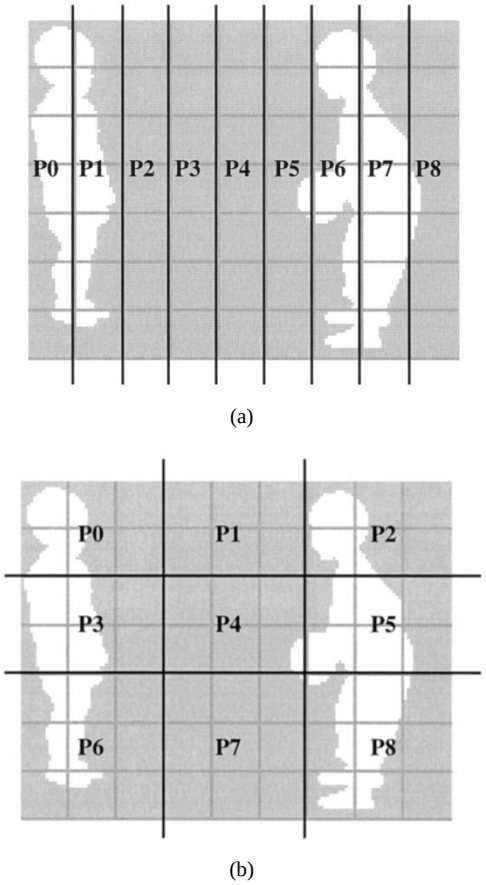


Fig. 10. Rectangular block partition example on "Children." (a) Strip-wise partitioning. (b) Block-wise partitioning.

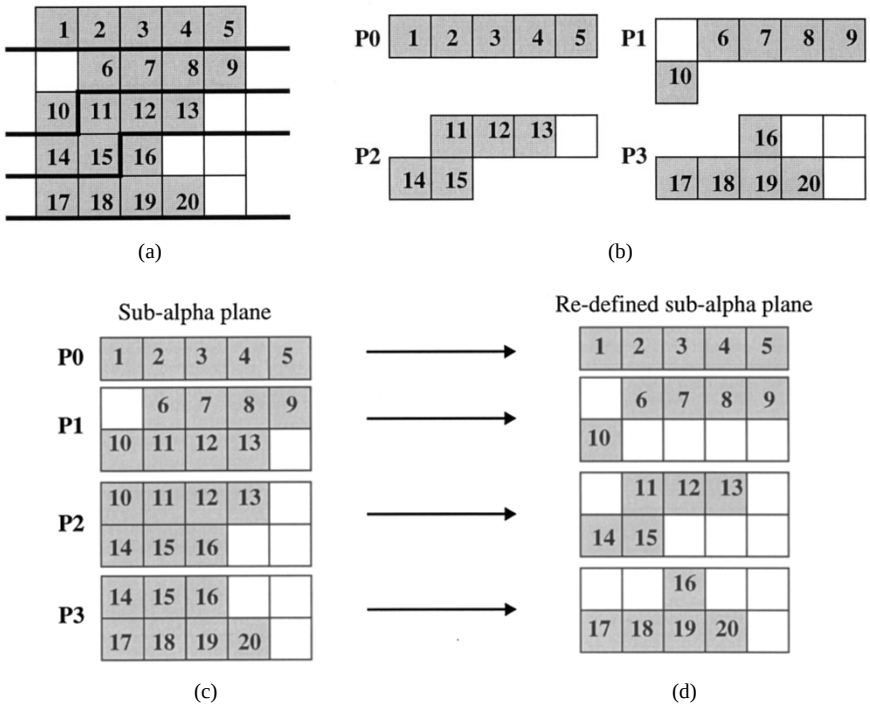


Fig. 11. Arbitrary partitioning example.

puting power), while other processors may be overloaded with computation-intensive contour and standard macroblocks.

In our shape-adaptive partitioning method, the initial separated subregion may have an arbitrary shape to minimize the imbalances. The extended rectangular subalpha plane is further redefined to avoid unnecessary computation for each processor.

As depicted in Fig. 11, gray blocks represent the contour and standard MB's, while white blocks represent the transparent MB's. By using the alpha plane information, first, we get the statistical distribution of the contour and standard MB's. Then, we equally distribute them to a given number of processors. As illustrated in Fig. 11(a), there are 20 contour and standard MB's within the window, and each processor is assigned five contour and standard MB's. Thus, each processor (P_0–P_3) may get an arbitrarily shaped subregion [see Fig. 11(b)]. As it stands, this partitioning will cause an irregular data structure problem. Moreover, because the bit stream can only indicate the rectangular data formation by the syntax such as *vop_horizontal/vertical_mc_spatial_ref* and *vop_width/height*, the decoding and picture composition will become more complex for arbitrary subregions. Here, we extend each subregion to the tightest rectangle called the subalpha plane [see Fig. 11(c)]. Since some of the subalpha planes contain redundant macroblocks, we redefine the subalpha planes by labeling those extensive macroblocks as transparent MB's in order to avoid unnecessary computation [see Fig. 11(d)]. For example, processor P_3 should only encode the subregion which includes the contour and standard macroblock from 16 to 20 as shown in Fig. 11(b). In order to get a rectangular subregion which contains those blocks, we extend this subregion to be a subalpha plane [Fig. 11(c)] which contains the contour and standard MB's from 14 to 20. Next, we define the fourteenth and fifteenth MB as transparent MB's to form a redefined subalpha plane as shown in Fig. 11(d). Therefore, processor P_3 still processes five contour and standard MB's while keeping the subregion rectangular. Because such a partition is based on macroblock decomposition, the granularity is small, and the method can yield finer workload balancing among the workstations. Fig. 12 compares the computation load distribution of each processors for the sequence "*Children*." It can be seen that both the stripwise and blockwise partitioning methods exhibit highly unbalanced distribution of computation load, while the shape-adaptive partitioning method achieves better load balancing.

IV. Experimental Results

With the advancement of workstations and networking technologies, the aggregated computing power of a cluster of workstations can approach that of an expensive parallel computing system [14]. Because of numerous advantages offered by this approach (scalable file storage, large memory, high performance/cost ratio, and efficient communication hardware/software support), many current parallel applications are now using clusters of workstations as the computing platform. The proposed parallel approach has been implemented on a cluster of 20 Sparc Ultra 1 workstations connected by a ForeSystems ATM switch (ASX-1000) which provides fast communication among the workstations. The cluster is configured as a virtual 2-D processor grid which is independent of the hardware topology.

For interprocessor communication and synchronization, we use the *message passing interface* (MPI), which ensures the portability of our MPEG-4 video encoder across various platforms. MPI is an industrial standard designed by the

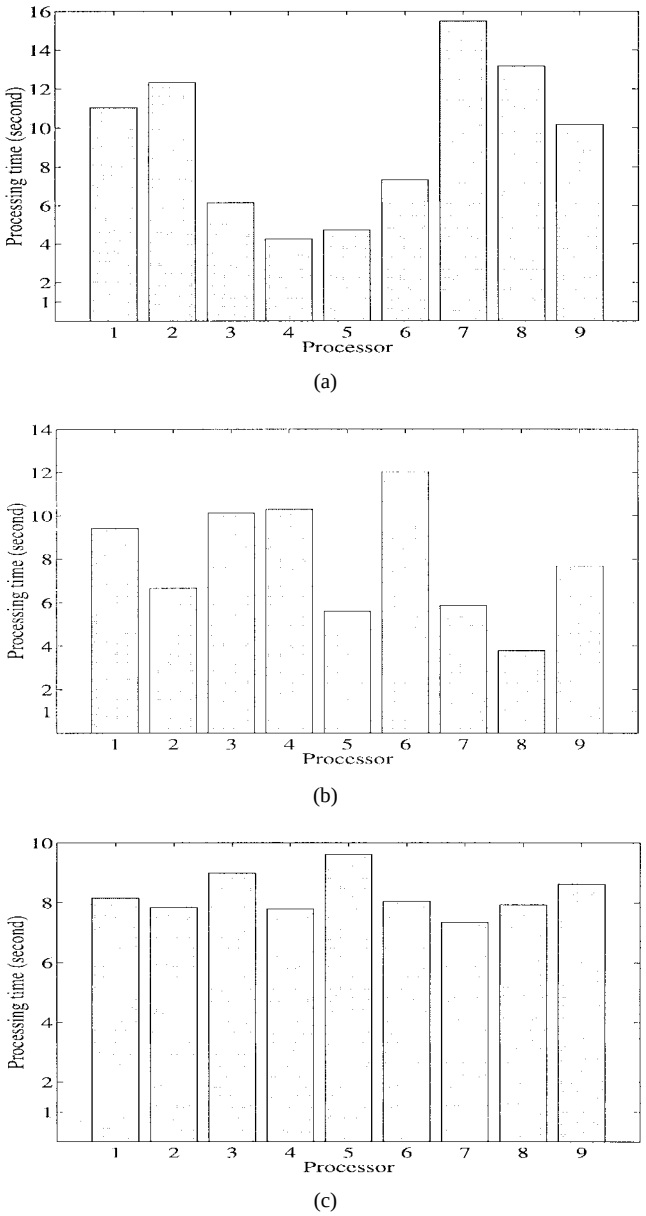


Fig. 12. Comparison of load distribution for different partitioning method. (a) Stripwise partitioning (1 × 9). (b) Blockwise partitioning (3 × 3). (c) Shape-adaptive partitioning.

MPI Forum for supporting a portable message-passing parallel program on massively parallel computers as well as networks of workstations [15].

Some encoding problems such as rate control and buffer regulation require the information for updating the quantizer, and such information has to be collected and broadcast among the workstations frequently. This may become a bottleneck in parallel implementation when the number of processor increases. In our implementation, because the shape-adaptive data-partitioning scheme ensures that each processor owns equal numbers of contour and standard macroblocks which consume most available coding-bits budget, we can assign the bit quota or buffers equally to the processors within the group. By so doing, each processor can adjust its local subbuffer independently, and no data piping is required. For example, if four processors deal with one VOP at a bit rate of 64 000 bits/s, each processor encodes the quarter-VOP at a bit rate of 16 000 bits/s.

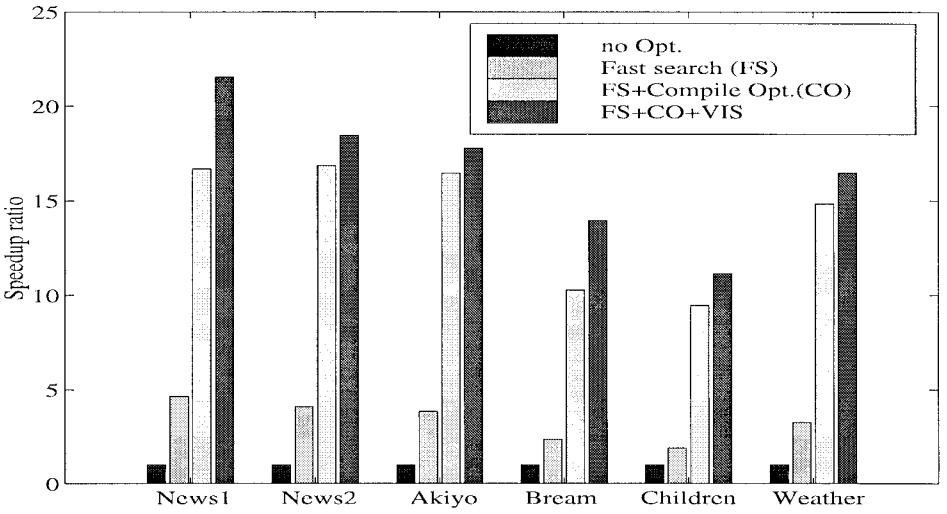
TABLE II
VIDEO OBJECTS FOR EXPERIMENTS

Sequence	Video object
News1	Dancers in the monitor
News2	Spokespersons
Akiyo	Woman
Bream	Fish
Children	Kids
Weather	Woman

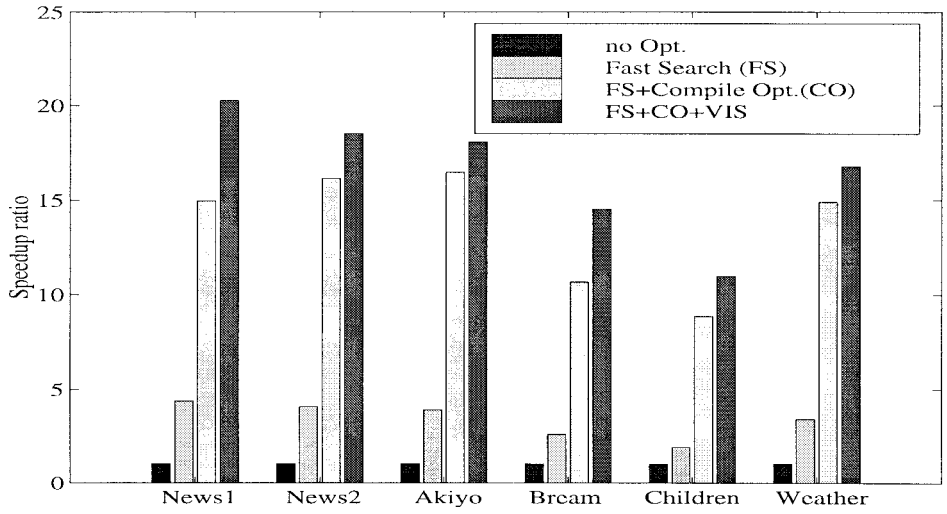
Several experiments have been performed on a set of MPEG-4 video test sequences (both single and multiple objects). Table II shows various single objects in different video sequences chosen from different classes of an MPEG-4 test library and represent various characteristics in terms of spatial detail and movement. Our software-based implementation is applied on the MPEG-4 video verification model (VM8.0) encoder.

We have also done software optimizations at various levels, which results in a significant performance improvement of the encoding speed. First, a fast search algorithm [16] has been adopted to speed up the computation of block-matching motion estimation while maintaining the visual quality close to that of the full search. We partition the search range into three nested search zones, and set the threshold to ten (zero means full search) for all zones. Second, we have used a Sun Solaris C compiler (SC4.0) to incorporate various compile optimizations. The appropriate option choice (such as *xarch, xchip, fast,* and *XO5)* can provide executable optimizations on the specific architecture, including loop parallelization and restructuring, cache properties redefinition, automatic register allocation, etc. Finally, we have utilized the visual instruction set (VIS) to compute the SAD, which is the most computational-intensive module in the encoding. VIS is a RISC-like extension to the SPARC V9 instruction set for the acceleration of multimedia processing on UltraSPARC processors [17]. By using the 64-bit floating-point register, VIS can perform the computations such as addition, subtraction, and multiplication on eight 8-bit pixels in a single cycle, which results in nearly eight-fold speed-up gain. Fig. 13 presents the encoding speed up with or without optimization approaches using one workstation.

In order to reduce the interprocessor communication overhead during the motion estimation, we have used an overlapped data allocation method [18]. This method stores the entire search window data on local disk, and allows each processor to perform motion estimation independently without any data exchange. Usually, workstations have enough memory for storage, and we can evenly distribute the entire frame data to each workstation. Therefore, each workstation can access the data from its local memory, and perform the distributed load balancing more efficiently. All of the preprocessing, including the format conversion and bounding, are done off line



(a)



(b)

Fig. 13. Encoding speed-up comparison (with/without optimization). (a) Encoding speed up of the QCIF sequences. (b) Encoding speed up of the CIF sequences.

TABLE III
PROCESSOR DISTRIBUTION TO EACH VIDEO OBJECT

Object	Object size	Size ratio	4 procs.	8 procs.	12 procs.	16 procs.	20 procs.
News1	30 MBs	1	1	3	4	6	7
News2	52 MBs	1.73	3	5	8	10	13

As for the single object sequences, we have to guarantee playout time constraints and allocate the VOP data proportionally to each workstation. Fig. 14(a) and (b) shows the encoding frame rates for different MPEG-4 video test sequences with QCIF and CIF format using various numbers of workstations. We can achieve a frame rate much higher than the real-time performance (30 frames/s) on all QCIF format sequences. For CIF format sequences, since the object size is larger, a frame rate above 15 frames/s can be obtained.

Fig. 15(a) and (b) shows the overall speed ups for the QCIF and CIF format, respectively. A linear speed-up relationship demonstrates that the performance of the encoder can scale according to the number of workstations used.

Apart from the single-object sequences, we also tested our encoder with the composed sequence "News" which contains the object "*News1*" and "*News2*" in QCIF format. "*News1*" has 150 frames and "*News2*" has 300 frames. During the first 150 frames, "*News1*" and "*News2*" are synchronized to each other with a frame rate of 30 frames/s. Then, the object "*News1*" ceases, and "*News2*" continues to the end with the same frame rate. Table III indicates the processor allocation during the first 150 frames. The number of processors assigned to each

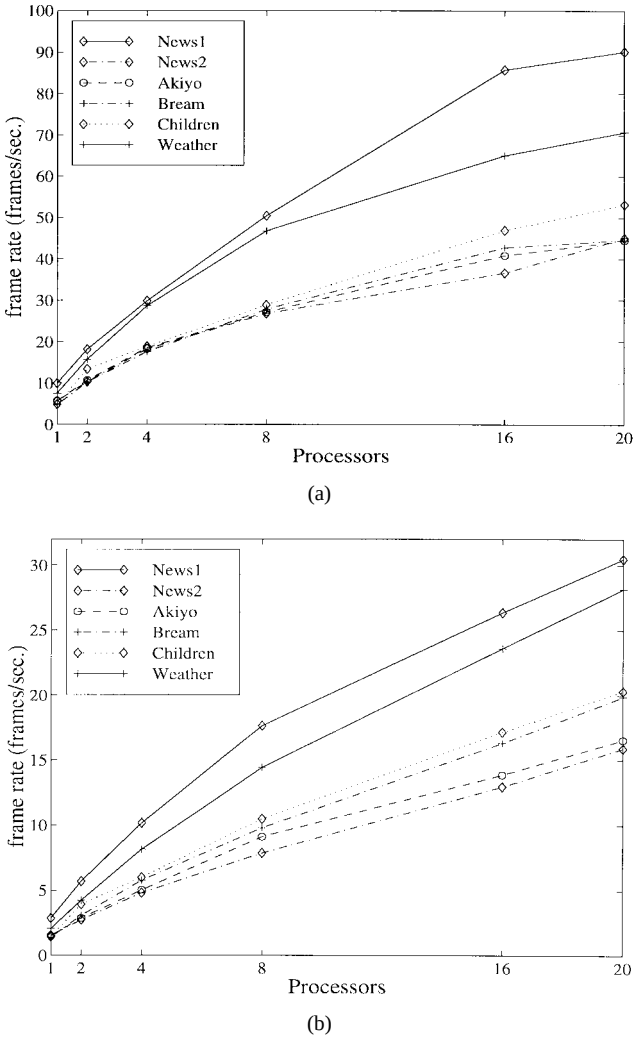


Fig. 14. Encoding frame rate. (a) QCIF format. (b) CIF format.

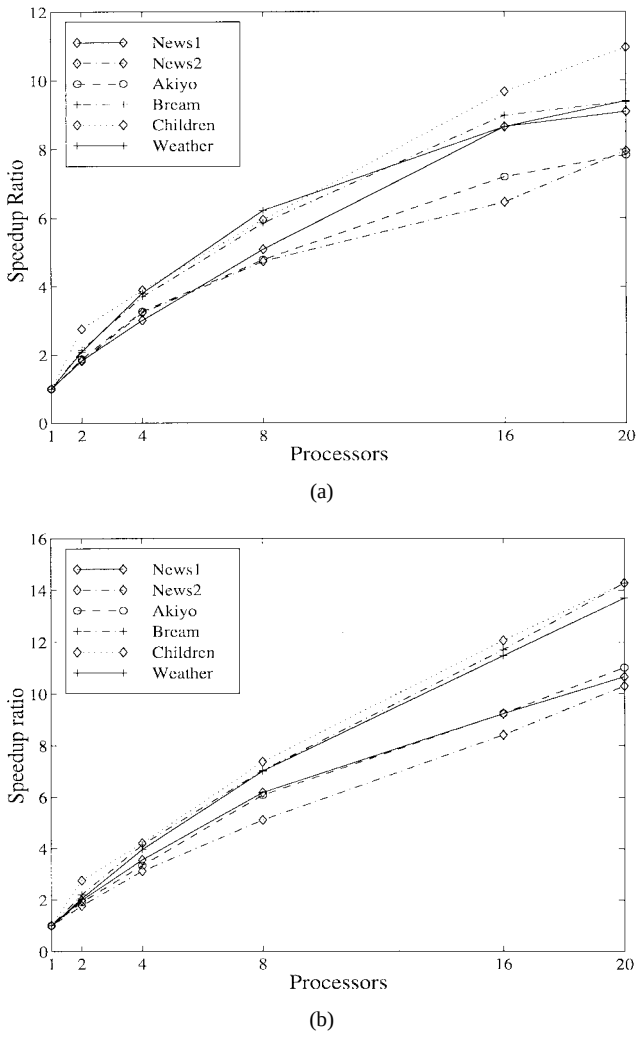


Fig. 15. Overall speed-up ratio. (a) QCIF format. (b) CIF format.

object is determined by the VOP window size ratio among the existing objects.

Fig. 16 is the encoding frame rate of "*News*" sequences with two objects inside the video session. The maximum encoding rate achieved on 20 workstations is about 41.78 frames/s.

V. Conclusions

In this paper a software-based MPEG-4 video encoder using parallel processing on a cluster of workstations has been described. The experimental results on various test sequences have been provided, and an encoding rate higher than real time has been achieved on some sequences. The contribution of our work includes the use of a hierarchical Petri nets model to capture the spatiotemporal relations between multiple objects of MPEG-4 video, an effective scheduling algorithm, and a dynamic shape-adaptive data parallel scheme to implement an MPEG-4 video encoder. In our future work, we will be exploring more efficient partitioning schemes and scheduling algorithms to improve the encoder performance. To complete our MPEG-4 multimedia system, we are also implementing MPEG-4 decoder and interactive methodology for supporting multimedia communication between the encoder and decoder.

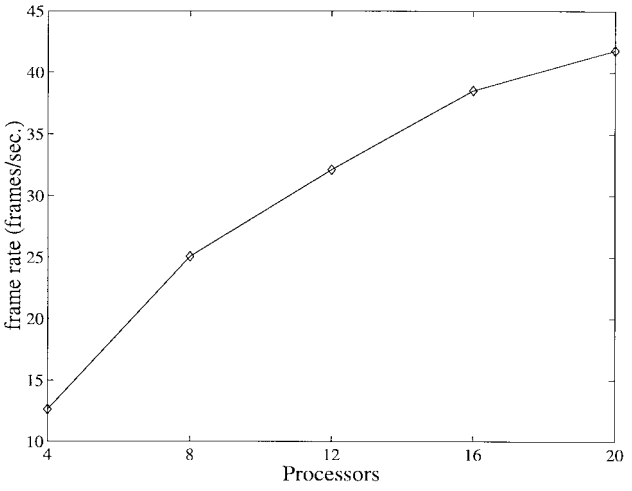


Fig. 16. Encoding frame rate of composed sequence "News" (QCIF).

Acknowledgment

The authors would like to thank Dr. R. Yung of Sun Microsystems and Dr. Y.-Q. Zhang of Sarnoff Corporation for technical support.

REFERENCES

[1] ISO/IEC, "Overview of MPEG-4 version 1 standard," ISO/IEC JTC1/SC29/WG11 N1909, Oct. 1997.

[2] T. Sikora, "The MPEG-4 video standard verification model," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 7, pp. 19–31, Feb. 1997.

[3] ISO/IEC, "MPEG-4 video verification model version 8.0," ISO/IEC JTC1/SC29/WG11 N1796, July 1997.

[4] G. Blakowski and R. Steinmetz, "A media synchronization survey: Reference model, specification and case studies," *IEEE J. Select. Areas Commun.*, vol. 14, pp. 5–35, Jan. 1996.

[5] T. Murata, "Petri nets: Properties, analysis and applications," *Proc. IEEE*, vol. 77, pp. 541–580, Apr. 1989.

[6] T. D. C. Little and A. Ghafoor, "Synchronization and storage models for multimedia objects," *IEEE J. Select. Areas Commun.*, vol. 8, pp. 413–427, Apr. 1990.

[7] M. Woo, N. U. Qazi, and A. Ghafoor, "A synchronization framework for communication of pre-orchestrated multimedia information," *IEEE Network*, vol. 8, pp. 52–61, Jan.–Feb. 1994.

[8] M. Diza and P. Senac, "Time stream Petri nets: A model for multimedia streams synchronization," in *Proc. 1st Int. Conf. Multi-Media Modeling*, Singapore, 1993, pp. 257–273.

[9] Y. K. Kwok and I. Ahmad, "Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors," *IEEE Trans. Parallel Distrib. Syst.*, vol. 7, pp. 506–521, May 1996.

[10] S. Cheng *et al.*, "Scheduling algorithms for hard-real time systems—A brief survey," in *Hard Real-Time Systems.* Los Alamitos, CA: IEEE Computer Society Press, 1988.

[11] J. M. Sohn and G. Y. Kim, "Earliest-deadline-first scheduling on nonpreemptive real-time threads for a continuous-media server," in *Proc. High-Performance Computing and Networking, Int. Conf. Exhibition*, 1997, pp. 950–956.

[12] I. Ahmad, "Resource management of parallel and distributed systems with static scheduling: Challenges, solutions and new problems," *Concurrency: Practice Exp.*, vol. 7, pp. 339–348, Aug. 1995.

[13] M. J. Berger and S. H. Bokhari, "A partitioning strategy for nonuniform problems on multiprocessors," *IEEE Trans. Comput.*, vol. C-36, pp. 570–580, May 1987.

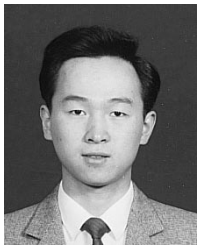
[14] T. E. Anderson, D. E. Culler, and D. Patterson, "A case for NOW (networks of workstations)," *IEEE Micro*, vol. 15, pp. 54–64, Feb. 1995.

[15] D. W. Walker and J. J. Dongarra, "MPI: A standard message passing interface," *Supercomputer*, vol. 12, pp. 56–68, Jan. 1996.

[16] Z. L. He and M. L. Liou, "A high performance fast search algorithm for block matching motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 7, pp. 826–828, Oct. 1997.

[17] M. Tremblay, J. M. O'Connor, V. Narayanan, and L. He, "VIS speeds new media processing," *IEEE Micro*, vol. 16, pp. 10–20, Aug. 1996.

[18] S. M. Akramullah, I. Ahmad, and M. L. Liou, "A software-based H.263 video encoder using a cluster of workstations," *Proc. SPIE*, vol. 3166, pp. 266–273, 1997.



Yong He (S'96) was born in Shanghai, China. He received the B.Eng. and M.Eng. degrees from the Southeast University, NanJing, China, in 1992 and 1995, respectively. He is currently a Ph.D. candidate in the Department of Electrical and Electronic Engineering, at the Hong Kong University of Science and Technology.

His research interests include image processing, video coding techniques, parallel and distributed computing, and scheduling algorithms.

Ishfaq Ahmad (S'88–M'91) received the B.Sc. degree in electrical engineering from the University of Engineering and Technology, Lahore, Pakistan in 1985. He received the M.S. degree in computer engineering, and the Ph.D. degree in computer science, from Syracuse University, in 1987 and 1992, respectively.

Currently, he is an Associate Professor in the Department of Computer Science at the Hong Kong University of Science and Technology. His research interests include various aspects of parallel and distributed computing, high-performance computer architecture and their assessment, multimedia systems, and video coding. He has published extensively in the above areas.

Dr. Ahmad received the Best Student Paper Awards at Supercomputing '90 and Supercomputing '91. He has been a guest editor for two special issues of *Concurrency: Practice and Experience*, and is guest editing a forthcoming special of the *Journal of Parallel and Distributed Computing*. He has also served on the program committees of various international conferences. Dr. Ahmad is a member of the IEEE Computer Society.

Ming L. Liou (M'63–SM'78–F'79) received the B.S. degree from National Taiwan University, the M.S. degree from Drexel University, Philadelphia, PA, and the 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 Hong Kong 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 parallel and distributed computing systems for visual applications. 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 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. He has published numerous papers in various fields.

Dr. Liou received the IEEE CAS Society Special Prize Paper Award in 1973 and the Darlington Prize 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 1979–1981, Executive Vice-President of CAS Society in 1986 responsible for regional activities, President of the CAS Society in 1988, the founding editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY from 1991 to 1995, and general Co-Chair of the 1997 IEEE International Symposium on Circuits and Systems held in Hong Kong. He is a member of Sigma Xi, Eta Kappa Nu, Phi Tau Phi, the Hong Kong Institution of Science, and a Fellow of the Hong Kong Institution of Engineers.