Available online at www.sciencedirect.com
SCIENCE DIRECT®
Parallel Computing 30 (2004) 1233–1276
PARALLEL COMPUTING
www.elsevier.com/locate/parco

Parallel parsing of MPEG video on a shared-memory symmetric multiprocessor

Suchendra M. Bhandarkar *, Shankar R. Chandrasekaran

Department of Computer Science, 415 Boyd Graduate Studies Research Center, The University of Georgia, Athens, GA 30602-7404, USA

Received 1 July 2002; accepted 21 May 2004

Abstract

Video parsing refers to the detection and classification of abrupt and gradual scene changes in a video stream. The detection of these changes forms an important preprocessing step in applications that treat videos as sources of information. The parsed video is subsequently indexed to support content-based retrieval, navigation and browsing. Analysis of video streams is computationally intensive with high data processing bandwidth requirements. Shared-memory symmetric multiprocessors (SMPs) have become increasingly ubiquitous and affordable. Parallel processing on an shared-memory symmetric multiprocessor is hence proposed as a means of dealing with the computational demands of video parsing. Parallel versions of two algorithms that detect scene transitions in compressed video streams are proposed. Both algorithms entail minimal decompression of the MPEG video. Three granularities of parallelism based on data decomposition and task decomposition are investigated; Group of Pictures (GOP), Frame and Slice. Results of an SMP implementation show that the GOP-level implementation, which represents the coarsest granularity of task and data decomposition, performs the best in terms of speedup and synchronization overhead. The slice and frame levels of granularity take second and third place, respectively. The speedup is observed to be almost linear in the case of the GOP level of granularity, whereas the

* Corresponding author. Tel.: +1 706 542 1082; fax: +1 706 542 2966.
E-mail address: suchi@cs.uga.edu (S.M. Bhandarkar).

0167-8191/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.parco.2004.05.002

synchronization overheads are observed to be high for the frame and slice levels of granularity.
© 2004 Elsevier B.V. All rights reserved.

Keywords: MPEG video; Video analysis; Video segmentation; Shared memory symmetric multiprocessors; Parallel video parsing

1. Introduction

With the emergence of *multimedia information systems*, videos are being increasingly looked upon as vital sources of information. A multimedia information system goes well beyond a traditional information system in that it incorporates various modes of non-textual digital data, such as digitized images, video and audio. One of the greatest problems with emerging multimedia technologies is the difficulty of rapidly and reliably extracting ''key'' information from images, video and audio streams which could then be used for rapid browsing, indexing and content-based retrieval of the relevant information [5,6,53]. Prominent applications that benefit from this research are video/multimedia database servers, video surveillance and digital libraries.

A great deal of current research effort has been devoted to automatic extraction of relevant information from video streams. Video parsing or scene change detection in a video stream is typically used to extract key features in a video stream. These key features are then used for rapid video browsing and automatic annotation and indexing of video streams to support content-based access to large video databases. The video parsing operation is primarily domain-independent, i.e., no assumptions are made about the semantics of the video or its underlying theme. Video parsing, therefore, is a crucial first step that precedes domain-dependent analysis of the video [53]. Some of the domain-independent features of interest in a video stream include scene cuts, scene dissolves, fade-ins, fade-outs, pans and zooms. A scene cut is an abrupt scene change. Fig. 1 depicts a scene cut between frames 89 and 90 in the *Table Tennis* video sequence. Pans, dissolves, fades and zooms represent gradual scene changes in

(a) Frame 88 **(b)** Frame 89 **(c)** Frame 90

Fig. 1. Scene cut between frames 89 and 90 in the *Table Tennis* video sequence. (a) Frame 88, (b) Frame 89 and (c) Frame 90.

the video stream. Figs. 2–4 show the video frames involved in a pan, dissolve and zoom respectively, in a video stream.

Due to the large amount of data involved, video streams are often compressed for efficient transmission and storage. Scene detection techniques that are capable of operating on compressed video data directly have a considerable advantage in terms of execution time and memory requirement when compared to those that require full frame decompression. In this paper we focus on detecting scene changes in compressed video streams that are encoded using the MPEG-1 standard [9,14,30,31,37].

The MPEG-1 video compression algorithm relies on two basic techniques: block-based motion compensation for reduction of temporal redundancy and Discrete Cosine Transform (DCT)-based compression [48] for the reduction of spatial redundancy. The motion information is computed using 16×16 pixel blocks and is transmitted along with the spatial information. The motion information is also compressed for higher efficiency. The temporal redundancy of video signals is exploited using motion compensated prediction which assumes that the current frame can be *locally* modeled as a transition of a video frame in the past or future. The MPEG-1 standard defines three types of frames:

Fig. 2. Sequence of frames showing a pan.

Fig. 3. Sequence of frames in a video stream spanning a dissolve.

Fig. 4. Sequence of frames in a video stream spanning a zoom-in.

- **I frame** (Intracoded): These pictures are purely intracoded using a DCT-based technique [48] and yield the least compression. They are used mainly as random access points in the video stream.
- **P frame** (Predictive): These frames are encoded using forward predictive coding, i.e., with respect to a previous I or P frame. These frames give better compression than I frames.
- **B frame** (Bidirectional or Interpolative): These frames are encoded using past and/or future I/P frames as reference frames. B frames achieve the highest compression but random access using these frames is not possible since they depend on a future and/or past reference frame.

The basic unit for motion compensation in MPEG-1, is a 16×16 block of pixels called a macroblock. Each frame is divided into a fixed number of macroblocks. The motion estimation for encoding P and B frames entails finding, for each macroblock in the P or B frame, the closest matching macroblock in the reference frames. In an I frame, each macroblock is intracoded, using the DCT. In a P frame, a macroblock may be intracoded or forward predicted. In a B frame, each macroblock may be intracoded, forward predicted, backward predicted or bidirectionally coded. The encoder performs the function of calculating appropriate motion vectors for each macroblock. The difference signal (prediction error) is also compressed using spatial redundancy reduction techniques based on the DCT. The DCT-coded prediction error signal consists of 16×16 pixel blocks and is transmitted along the rest of the spatial information. Therefore, depending on the type of macroblock, motion vector information is stored along with the compressed prediction error signal in each macroblock.

To exploit spatial redundancy, the MPEG-1 standard uses block-based DCT encoding with weighted quantization and run-length encoding [48]. After motion compensation is performed, the remainder of data is divided into 8×8 blocks, DCT encoded, weighted, quantized and run-length encoded for efficiency. During the decoding of the MPEG-1 stream, I frames can be decoded independently. Decoding of P frames requires the previously decoded reference frame (I or P) whereas decoding of B frames requires either 1 or 2 reference frames (from the past or future). So typical decoders buffer up to 3 frames (2 reference frames and the frame being currently processed) in memory at any point in time. To facilitate decoding, encoders typically rearrange the frames (out of their temporal sequence) in the MPEG-1 stream so that the reference frames are already decoded before they are needed for decoding a B frame.

An MPEG-1 video stream is hierarchically structured. Each stream consists of one or more sequences. A sequence is a random access unit and could represent a scene context. A sequence consists of one or more Groups of Pictures (GOP). A GOP consists of one or more frames. Each GOP starts with an I or a B frame and ends with an I or a P frame. A frame, as discussed previously, can be encoded as an I, P or B frame. A frame consists of one or more slices where a slice is used to resynchronize the video stream in case of errors. A slice comprises one or more macroblocks. A macroblock can be decomposed into pixel-level blocks.

A macroblock is the 16×16 motion compensation unit whereas a block is the 8×8 DCT unit.

Several MPEG-1 video parsing algorithms are based on the analysis of *DC images*. DC images are spatially reduced versions of the original frames in the MPEG-1 stream and represent the DC coefficients of the DCT (Fig. 5). A video sequence comprised of DC images is called a DC sequence. A DC image retains most of the global information in the original frame while requiring only a small fraction of the original frame memory. Processing of DC images is much faster and the results are often satisfactory. MPEG-1 uses the YUV color format and each frame in the video stream is encoded as three frames which are termed Y, U and V frames. The Y frame represents the brightness or luminance component of the original frame and U and V frames represent the color or chrominance components. A DC image of a frame comprises DC images corresponding to the Y, U and V frames.

Traditionally parallel processing has been used for MPEG encoding and decoding since both, encoding and real-time decoding are computationally intensive. Both, shared and distributed memory architectures have been used for parallel video processing. To date, we are not aware of any published work on parallel MPEG-1 video parsing. As parsing is closely related to decoding, it can also be expected to benefit from parallelism. MPEG-1 coding with its inherent dependencies and other nuances poses several challenges to the parallelization of video parsing.

The algorithms discussed in this paper directly exploit the motion compensation, the prediction error signal and the DC coefficients in the DCT encoding of the MPEG-1 video stream. The algorithms entail minimal decoding of the compressed video stream, thus resulting in significant savings in terms of execution time and memory requirement. The parallel algorithms are implemented on a shared memory symmetric multiprocessor (SMP). We use the shared memory model for our implementation on account of its relative ease of programming abstraction and faster realization. For a small number of processors, parallel implementations on shared memory systems are typically faster than on distributed memory systems since the interconnection network latency is observed to be a performance limiting factor in the case of the latter.

Fig. 5. Original frame from the *Table Tennis* video sequence and its DC image. (a) Original frame and (b) DC image.

2. Related work

There has been a great deal of work on video parsing or scene change detection on uncompressed video. Most of these techniques compute a global measure of difference between successive frames or images in the video stream and use a global thresholding scheme to determine and localize significant scene changes in the video stream. The difference measures are typically based on gray level sums (computed over the entire frame), gray level histograms, color histograms, gray level statistics or color statistics computed for each frame in the video stream [1,32,34,39,52,53], or motion discontinuities computed using temporal filtering [18,17,41]. Feature-based approaches compute the difference in features such as edges between successive frames of the uncompressed video [51]. In some cases, a domain-specific video production model is used to guide the parsing process [11,15,47].

As previously mentioned, parsing techniques that entail minimal decompression of the video stream have a definite advantage in terms of run time and memory usage. A number of video parsing techniques are based on the analysis of DC images [24,25,42,43,49,50]. Shen et al. [42,43] suggest the use of histograms of the DC images derived from the DCT coefficients of the I frames in an MPEG-1 video stream to detect scene changes. Scene cuts are detected by thresholding the difference of the luminance histograms computed from the DC images of two consecutive I frames in the MPEG-1 video stream. Dissolves are detected by computing the histogram difference between the current DC image and the average of all DC images in a window preceding the current I frame. Patel and Sethi [35,36,40] use the DC coefficients of the I frames to perform hypothesis testing on the luminance histogram. Their approach implicitly assumes that the separation between successive I frames in an MPEG-coded video stream is fixed and small. Meng et al. [29] and Ching et al. [12] use the variance of the DC coefficients in the I and P frames and the proportion of macroblocks in the P and B frames that are intracoded to detect scene changes. Zhang et al. [54] and Kobla et al. [22,23] have observed that the number of macroblocks with valid motion vectors in P or B frames tends to be low when these frames lie on opposite sides of a scene change. This fact is used for detection of abrupt scene changes whereas more gradual scene transitions are detected after decompressing the relevant subset of video frames and analyzing the motion in the decompressed video stream. Koprinska and Carrato [24,25] use a hybrid rule-based/neural network approach to scene change detection where simpler scene boundaries are detected by the rule-based system and the complex scene boundaries are detected by the neural net. Song et al. [46] use a chromatic video editing model for low motion videos based on the observation that the first order partial temporal derivative of the video signal is zero for static scenes and the second partial temporal derivative is zero for scenes with gradual changes.

Gamaz and Huang [13] propose a simple skipping algorithm for fast and accurate detection of abrupt scene changes. The two I frames in a GOP are compared using DC differences. If there is no appreciable difference, that GOP is skipped, else the P and B frames in the GOP are processed for changes. The algorithm is simple and fast but does not work for complex scene changes. Shen and Sethi [44,45] describe algo-

rithms to detect features such as spatial gradients in MPEG-1 video which could be subsequently used to detect scene changes. Yeo and Liu [49] present algorithms for detecting abrupt and gradual scene changes, intrashot variations and flashlight scenes in MPEG and motion JPEG video streams [48] using DC images. They present an efficient technique for extracting a sequence of DC images from a compressed video stream [49]. The key frames derived from the scene change detection algorithm are subject to a clustering and matching process to generate a scene transition graph which can be used for high-level representation and rapid browsing and navigation of the video data [50]. Ngo et al. [33] use one dimensional spatio-temporal strips derived from DC images and reduce the problem of working with a sequence of frames to a problem of working with a two-dimensional image. A statistical DC histogramming approach is used to detect the abrupt and gradual scene changes.

Video parsing techniques that rely primarily on chrominance and/or luminance values (or their averages as reflected in the DC images) for the purpose of scene change detection are prone to misses when there is little change in background color or luminance between successive video shots and to false positives when there is a change in background color, luminance or ambient lighting within a single shot. Bhandarkar and Khombadia [7] have presented algorithms for computing the relative motion between successive frames to detect abrupt and gradual scene changes in a compressed video stream. However, the approach suffers from the drawback that it is not always possible to determine the motion vectors in the MPEG-1 video stream [7]. Consequently, Bhandarkar et al. [8] have presented parsing algorithms that use both DC images and motion vectors for detecting scene changes in MPEG-1 video. It was shown that an integrated approach, that uses both, motion vectors and DC histograms, performs better than the individual methods. Another class of techniques does video parsing at the level of scenes and exploits frame and shot similarity measures based on a semantic model. Kender and Yeo [21] present novel high-level approaches for segmenting hierarchical scene structure in video. Scene boundaries are determined using shot-to-shot coherence followed by a one pass on-the-fly shot clustering algorithm. The video segmentation is then done at the *theme* level.

The output of the parsed video is used in content-based retrieval of the video database [38]. Lienhart et al. [26] present a systematic method to compare and retrieve video sequences at four levels of temporal resolution, i.e., frame, shot, scene and video. The video is transformed to an appropriate representation before comparisons are made. A normalized measure of distance between the representations of two video sequences is defined for similarity. The method is domain independent and can compare frames, shots or sequences. Jain et al. [19] present a technique for querying a video database by content using video clips. The articles by Ahanger and Little [3] and Jiang et al. [20] give an exhaustive review of video parsing techniques and their use in content-based querying of video and multimedia databases.

MPEG-1 encoding and decoding have received the most attention from the viewpoint of parallel processing. Bilas at al. [10] have implemented a shared memory parallel MPEG-1 decoder. Their scheme consists of a scan process that scans the MPEG-1 stream, generates tasks and inserts them in a common queue. A pool of worker processes removes the tasks from the queue and processes them. Their

algorithm does not include frame level parallelism and assumes that all GOPs are closed (i.e., all the inter-frame references lie strictly within a GOP). Mayer-Patel [27] discusses parallel processing for special video effects such as titling, compositing and blending. Three modes of parallelism are considered: (a) functional parallelism where the video effects are decomposed into smaller subtasks that are mapped on individual processors, (b) temporal parallelism where the video stream is partitioned into sets of successive frames and the processors work on the entire set of frames assigned to them, and (c) spatial parallelism where regions of video frames are assigned to individual processors. For instance, the left halves of all the frames can be processed by one processor and the right halves can be processed by the other processor.

Shen et al. [42] have implemented an MPEG-1 encoder on the Intel Touchstone Delta and Intel Paragon parallel computers using the SPMD model of parallelism. Akramullah et al. [4] have implemented a parallel MPEG-2 encoder on a network of workstations connected via Ethernet and ATM. Parallelism is achieved at the level of the GOP, i.e., each GOP is encoded by a single processor. A scheme for efficient I/O and data distribution is presented. Although encoding is often done off line, the performance is shown to be better than real-time. He et al. [16], propose a software-based MPEG-4 encoder using a network of workstations. A Petrinet-based modeling methodology is used to capture spatio-temporal relationships among the multiple video objects at different levels of the MPEG-4 encoding hierarchy. Their scheme incorporates automatic partitioning, allocation and scheduling of video objects to individual processors as well as dynamic determination of execution order and synchronization requirements. With 20 processors, the performance is better than real-time and multiple sequences can be encoded simultaneously. Agi and Jagannathan [2] discuss a parallel MPEG-1 encoder on a CM-5 Machine where parallelism is achieved at the GOP level. Their scheme shows a linear speedup up to 16 processors, beyond which the speedup drops due to the bottlenecks in the communication network and the file I/O system. Meng et al. [28] present a compressed video searching and editing system called WebClip for the web. WebClip works on MPEG-1 compressed video allowing for editing functions such as cuts, pastes, blends, dissolves etc. and does task/data distribution at the level of the GOP.

3. Parallel parsing of MPEG-1 video

Two different approaches to parallel parsing of MPEG-1 video are implemented and analyzed in this paper. The first approach, termed as *Approach 1*, is based on [7,8] and uses DC images and motion vectors to detect scene changes. The DC images corresponding to Y, U and V images of each frame in the MPEG-1 stream are generated using the method suggested in [49,50] which does not entail complete decompression of the MPEG-1 stream. Abrupt scene changes (i.e., scene cuts) are detected by thresholding the difference between the sum of the DC values from the Y, U and V images of two consecutive frames. If there is a scene change between two frames, this difference will exceed a prespecified threshold. Gradual scene

changes (i.e., fades and dissolves) are detected by thresholding the difference between the sum of DC values that are k frames apart [49].

In the case of video parsing using motion vectors, we exploit the fact that scene changes such as scene cuts, pans and zooms exhibit definite patterns in the underlying motion vector field. Abrupt scene changes are found by first computing the *motion distances* between two successive frames [7]. The motion distance denotes the difference in the extent of motion between successive frames. Scene cuts are characterized by large motion distances between the successive frames on either side of the scene boundary [7]. In a pan scene, a majority of the motion vectors are aligned in the direction of the pan. During a zoom-in or zoom-out, a majority of the motion vectors either point outward or inward respectively. Dissolves are detected based on the observation that the prediction error values in the encoded macroblocks are high. Fig. 6 depicts an outline of a sequential algorithm for Approach 1.

The second approach, termed as *Approach 2*, is based on [33] in which only DC image strips are used. Three strips, namely, the horizontal, vertical and diagonal strips passing through the center of each DC image, are extracted for each frame in the video. The corresponding strips from each frame are stitched together to form a set of three 2-D images where each 2-D image corresponds to the horizontal, vertical or diagonal strips. Scene changes are detected by segmenting these images. The shapes of boundaries between the segments reveal the nature of the scene transitions. Abrupt scene cuts and gradual scene changes are detected in the same manner as the DC image approach [49]. Thresholding is performed on the difference sequences. For some of the MPEG-1 streams that were tested, this approach gives better results than the DC image approach. Detection of zoom-ins and zoom-outs is much more complex and hence not performed. Fig. 7 depicts an outline of a sequential algorithm for Approach 2.

The two approaches discussed above are parallelized in our parallel video parsing system depicted in Fig. 8. The parallel video parsing system is similar to the parallel

```
procedure Approach 1 - sequential
begin
    for all frames in the video sequence
    do
        Extract DC images
        Store sum of DC values
        Store motion vector information
    end
    for ( i = 1, i < Number of frames, i++ )
    do
        Compute the DC differences between frames i and i − 1
        Compute motion distances between frames i and i − 1
        Compute the kth difference between frames i and i + k for dissolves
        Compute the kth difference between frames i − k and i for dissolves
    end
end
```

Fig. 6. Outline of the sequential algorithm for Approach 1.

```
procedure Approach 2 - sequential
begin
    for all frames in the video sequence
    do
        Extract the horizontal, vertical and diagonal strips from DC images
        Find edge differences between strips i and i − 1
        Store the sum of DC values of all strips
    end
    for ( i = 1, i < Number of frames, i++ )
    do
        Compute the kth difference between frames i and i + k for dissolves
        Compute the kth difference between frames i − k and i for dissolves
    end
end
```

Fig. 7. Outline of the sequential algorithm for Approach 2.

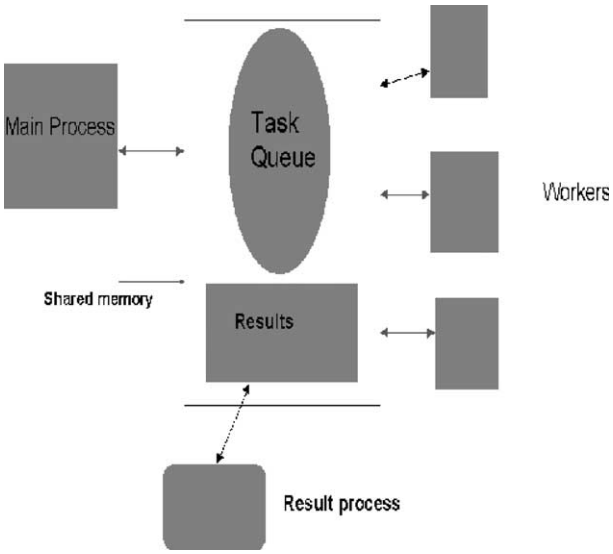

Fig. 8. System organization.

MPEG decoding system discussed in [10]. The system consists of a main or master process, a group of worker or slave processes and a display process. The master process scans the input MPEG-1 stream, prepares the tasks and assigns the tasks to the worker processes. The workers process the tasks and convey the intermediate and final results to the display process. The relevant information such as the DC images and the motion vectors are extracted locally within the tasks assigned to the workers and are shared with the other workers in order to compute the final

results. All the processes use shared memory for communication and synchronization. The master process prepares the parallel units of computation (i.e. tasks) and inserts them in a circular queue in the shared memory. The workers remove the tasks from the queue and process them. The master process does no computation per se, but monitors the MPEG-1 stream for the beginning and end of a new frame or a slice, or for a particular sequence of frames to build custom GOPs for the GOP-level implementation (as discussed in the next section) before queuing tasks in the task queue. All the decoding is done by the workers. The task queue is either common to all workers or each worker could have its own task queue.

Since the MPEG-1 stream can be hierarchically decomposed into various layers such as sequences, GOPs, frames, slices and macroblocks, it stands to reason to consider these layers as potential parallel units of work. We have chosen GOPs, frames and slices as possible units for task decomposition in our current implementation. The sequence layer is not chosen because many available MPEG-1 videos have very few sequences, typically just one. The macroblock layer is not chosen because in a given slice the macroblocks are highly dependent on other macroblocks. Thus task decomposition at the macroblock level would involve tremendous synchronization. As a passing note, in our system (as in most MPEG-1 decoders), only three frames and their related information reside in memory at any point in time. The three frames are the two reference frames and the frame being currently decoded or analyzed. The related information comprises of DC images, motion vectors, and other assorted data needed for the working of the system. Our parallel algorithms are implemented on a 32-processor SUN Enterprise 10000 server which is a shared-memory symmetric multiprocessor (SMP) running the SUN Solaris operating system. Interprocess communication is done solely through shared memory.

3.1. Parallel MPEG-1 video parsing at the GOP level

When parsing MPEG-1 video at the GOP level, each worker process tackles a distinct GOP and processes the results. There is a distinction between the GOP assigned to a worker process and the GOP present in the MPEG-1 stream. The traditional GOP in an MPEG-1 stream consists of a set of frames with a unique start code and header. As discussed in the introduction, a GOP in the MPEG-1 stream can start with an I or a B frame and end with an I or a P frame. A GOP header has a special bit called the closed bit. If this bit is set, no frame in that GOP depends on any other frame in the preceding or succeeding GOP. Such a GOP is considered closed and can be decoded independently. Thus, a closed GOP has to start with an I frame or a B frame with no dependence on the previous GOP. However, due to the fact that the MPEG-1 encoder transmits frames out of their true temporal order, the resulting GOPs are not closed (Fig. 9). GOPs that are not closed entail communication and synchronization overhead among the worker processes. Thus, the master process has to preprocess the actual MPEG-1 stream to ensure that the GOPs are always closed. The closed GOPs resulting from the preprocessing of the MPEG-1 stream are called custom GOPs (Fig. 9). In the remainder of the paper, the term GOP implies a custom GOP unless stated otherwise. Since the GOPs are independent, the

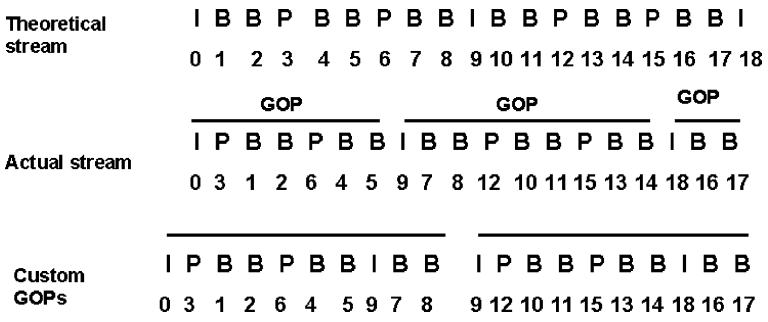

Fig. 9. The actual GOP in the MPEG-1 stream and the custom GOP.

communication and synchronization overhead among the worker processes is minimal. Most of the synchronization and communication is between the master process and the worker processes.

3.1.1. Parsing using DC images and motion vectors (Approach 1)

The master process inserts the custom GOPs along with relevant header information into the task queue. The worker processes remove the tasks from the queue and process them independently with no communication between the workers. Each worker process uses the algorithms in [7,8,49] for detecting scene cuts, pans, zooms and gradual scene changes using the DC images and motion vector information derived from the video segment within their respective GOPs. The results of the parsing are inserted in a single shared data structure in memory that is indexed by the frame number. No locks are required to access this shared structure since no two workers handle the same frame. The workers access the task queue until it is empty. Access to the task queue is synchronized. The pseudocode outlines for the master process and the worker process are shown in Figs. 10 and 11 respectively.

Arriving at the results from the GOPs takes place in two steps. In the first step, a worker process extracts the three DC images for each frame in its GOP, and computes the sum of the DC values in each of the 3 images. In the second step, the difference between the sum for the current frame and the sum for the previous frame is computed. Once the array of sums is available, finding the difference between sums of successive frames or between sums that are k frames apart [1] can be done in any manner. In the case where the span k of a dissolve is larger than the number of frames in a GOP, the difference computation requires the sum of the DC values of a frame in another GOP. This entails communication and synchronization among the workers. Computing the motion vectors is done in a similar manner. Instead of extracting the DC images, the workers compute the motion vectors within each GOP. Detection of scene cuts, pans and zooms is done by computing the motion vectors with respect to a reference frame within the GOP as is done in the sequential algorithm.

[1] For dissolve detection.

```
procedure Main Process - GOP
begin
    while Not end of MPEG stream
    do Scan stream for next I frame
        Construct custom-GOP
        if ( Task Queue == Full )
        then
            Wait
        Insert custom-GOP in task queue
        Signal workers
    end
end
```

Fig. 10. Pseudocode outline of the master process for parallel video parsing at the GOP level (Approach 1).

```
procedure Worker Process - Approach 1 , GOP-based
begin
    while Not end of stream
    do
        if ( Task Queue == Empty )
        then
            Wait
        Read custom-GOP
        for all frames in the custom-GOP
        do
            Extract DC images
            Store sum of DC values in shared memory
            Store motion vector information in shared memory
        end
        Signal master process
    end
    Barrier Synchronization
    Start = My first frame
    End = My last frame
    for ( i = Start, i < End, i++ )
    do
        Compute the DC differences between frames i and i − 1
        Compute motion distances between frames i and i − 1
        if ( kth sum exists )
        then
            Compute the kth difference between frames i and i + k for dissolves
        if ( (i − k)th sum does not exist )
        then
            Compute the kth difference between frames i − k and i for dissolves
    end
end
```

Fig. 11. Pseudocode outline of the worker process for parallel video parsing at the GOP level (Approach 1).

3.1.2. Parsing using DC image strips (Approach 2)

Recall that 1-D strips are extracted from the three DC images of each frame in the MPEG-1 stream and stitched together to form a 2-D image. Each worker locally

stitches the strips from the frames in its GOP. The scene changes are detected by segmenting the 2-D image using an edge detector in the (Y, U, V) feature space. Since edge detection involves finding the difference in values between DC images of consecutive frames, the worker processes need to synchronize at the GOP boundaries. The GOP-based scheme is modified slightly for this approach. Edge detection is done while extracting the strips. This approach saves memory since storing the entire 2-D image in memory is not necessary. Scene cuts and pans use the differences computed between consecutive frames. Dissolves are implemented using the same k-difference method used with the DC images in Approach 1. Each worker computes the sum of the strips and if the value of the kth strip exists within the process, it computes the difference. The sums obtained from the strips are kept in a shared address space. The difference is then thresholded to check for peaks and plateaus. As before, parallelism is affected when detecting dissolves if the span of the dissolve is greater than the number of frames in a GOP. The master process for Approach 2 is the same as that for Approach 1. The pseudocode outline for the worker process in Approach 2 is depicted in Fig. 12.

```
procedure Worker Process - Approach 2 , GOP-based
begin
    while Not end of stream
    do
        if ( Task Queue == Empty )
        then
            Wait
        Read custom-GOP
        for all frames in the custom-GOP
        do
            Extract the horizontal, vertical and diagonal strips from DC images
            Find edge differences between strips i and i − 1
            Store the sum of DC values of all strips in shared memory
        end
        Signal master process
    end
    Barrier Synchronization
    Start = My first frame
    End = My last frame
    for ( i =Start, i < End, i++ )
    do
        if ( kth sum exists )
        then
            Compute the kth difference between frames i and i + k for dissolves
        if ( (i − k)th sum does not exist )
        then
            Compute the kth difference between frames i − k and i for dissolves
    end
end
```

Fig. 12. Pseudocode outline of the worker process for parallel video parsing at the GOP-level (Approach 2).

3.2. Parallel MPEG-1 video parsing at the frame level

When parsing MPEG-1 video in parallel at the frame level, each worker process deals with a single frame at a time and computes the results. This entails finer-grained task and data decomposition compared to parallel video parsing at the GOP level. Since the results are based on differences between successive frames, worker processes dealing with successive frames need to synchronize and communicate their results. To minimize the synchronization communication overhead, intermediate results are temporarily buffered and the final results computed only after every I frame in the MPEG-1 stream. This permits the workers to compute the intermediate results locally without synchronization. But the inherent dependencies between I, P and B frames limits the extent of parallelism. If a worker is processing an I or a P frame, no other worker is allowed to proceed concurrently because the frames that follow the I or P frame will either depend on the I or P frame, or on a frame succeeding the I or P frame. Thus, the presence of the reference frames (I or P frames) in the MPEG-1 stream enforces sequential execution. Only a worker that is currently processing a B frame can execute concurrently with other workers that are also currently processing B frames. A higher percentage of B frames in the MPEG-1 stream increases the extent of parallelism that can be realized at the frame level.

3.2.1. Parsing using DC images and motion vectors (Approach 1)

The master process scans the MPEG-1 stream for the start and end markers of a frame and inserts the frame along with the appropriate header in the task queue. The worker processes remove the tasks from the task queue. Processing takes place in the order of the frame numbers due to the inter-frame dependencies. The frames that appear early in the MPEG-1 stream have to be processed before the later frames. Thus, the worker in possession of the frame with the lowest number takes precedence. If the frame is a reference frame (I or P frame), the frame number decides the order. If the frame is a B frame, then both of its reference frames must have been already

```
procedure Main Process - Frame
begin
    while Not end of MPEG stream
    do
        Scan stream for next frame start code
        Read frame
        if ( Task Queue == Full )
        then
            Wait
        Insert frame and corresponding GOP header in task queue
        Signal workers
    end
end
```

Fig. 13. Pseudocode outline of the master process for parallel video parsing at the frame level (Approach 1).

processed. The above two conditions enforce a global ordering on the computation. A worker, not processing a frame, sleeps on a condition variable. The pseudocode outline of the master process is given in Fig. 13.

The worker processes compute the results in two steps. In the first step, a worker process extracts the DC images for all of its assigned frames, computes all the necessary information from each frame including the sum of DC values and records the

```
procedure Worker Process - Approach 1, Frame-based
begin
    while Not end of stream
    do
        if ( Task Queue == Empty )
        then
            Wait
        Read frame
        if ( Last frame == I frame )
        then
            PreProcess
        while ( My frame NOT lowest )
        do
            Wait
        end
        if ( My frame == I frame )
        then
            Signal all workers
            PreProcess
        Extract DC images
        Store sum of DC values in shared memory
        Store motion vector information in shared memory
        if ( kth sum exists )
        then
            Compute the kth difference between frames i and i + k for dissolves
        if ( (i − k)th sum does not exist )
        then
            Compute the kth difference between frames i − k and i for dissolves
        Signal workers and master process
    end
end


procedure PreProcess , used by Worker process in Approach 1, Frame-based
begin
    Barrier Synchronization
    for all stored frames
    do
        Compute the DC differences between frames i and i − 1
        Compute motion distances between frames i and i − 1
    end
    Barrier Synchronization
end
```

Fig. 14. Pseudocode outline of the worker process for parallel video parsing at the frame level (Approach 1).

information in the shared memory. The first step is local to each worker process and does not entail any synchronization. The second step involves synchronization after every I frame in the MPEG-1 stream. After every I frame, the workers use the intermediate information to compute the final results. To make this possible, the worker that processes an I frame, sets a global variable that acts as a signal to all the other workers. During the processing of the I frame, all the other workers suspend their tasks since the I frame is a reference frame. After having processed its I frame, the worker awakens the other sleeping workers. Since the global variable is now set, all the workers halt the processing of any future frames and proceed to compute the final results. They divide the preprocessed work among themselves and compute the inter-frame differences. The pseudocode outline of the worker process for Approach 1 is shown in Fig. 14. The extraction of motion vectors is performed in a similar manner. As in the previous case, the reference frames are resident in the shared memory. To compute the motion distance for detection of scene cuts, the workers access the reference frames synchronously. Detection of zooms and pans also entails access to the reference frames [8]. Dissolves are detected using the prediction errors. The differences in the prediction errors between consecutive frames are thresholded to detect dissolves [7].

3.2.2. Parsing using DC image strips (Approach 2)

The master process for Approach 2 is the same as that for Approach 1. The worker processes extract the three strips, horizontal, vertical and diagonal, from the three

```
procedure Worker Process - Approach 2, Frame-based
begin
    while Not end of stream
    do
        if ( Task Queue == Empty )
        then
            Wait
        Read frame
        while ( My frame NOT lowest )
        do
            Wait
        end
        Extract the horizontal, vertical and diagonal strips from DC images
        Find edge differences between strips i and i − 1
        Store the sum of DC values of all strips in shared memory
        if ( kth sum exists )
        then
            Compute the kth difference between frames i and i + k
        if ( (i − k)th sum does not exist )
        then
            Compute the kth difference between frames i − k and i
        Signal workers and master process
    end
end
```

Fig. 15. Pseudocode outline of the worker process for parallel video parsing at the frame level (Approach 2).

DC images of the frames they are currently processing. The processing of the frames follows the same order as in the case of Approach 1. At the end of each frame, the worker processes synchronize to compute the pixel differences. Each worker process computes the differences between its current frame and the stored values of the previous frame and stores the current values in the shared memory for the worker processing the next frame in order. Dissolves are implemented using the same k-difference method as in the case of Approach 1 (with DC images). At the end of a frame, the worker computes the k-difference if the value of the kth frame exists locally. If the value does not exist locally, then the worker processing the kth frame computes the k-difference. The k-difference is then thresholded to check for peaks and plateaus for potential dissolves. The pseudocode outline of the worker process for Approach 2 is given in Fig. 15.

3.3. Parallel MPEG-1 video parsing at the slice level

When parsing MPEG-1 video at the level of slices, each worker is assigned a slice to be processed. This represents the finest-grained task and data decomposition compared to parallel video parsing at the GOP level or the frame level. The slice-level implementation has the same overall structure as the frame-level implementation. The workers process all the slices in a given frame before proceeding to the next frame. When a reference frame is being processed by a group of workers, no other worker can process the slices from the next frame. Likewise, when non-reference frames are processed by a group of workers, no other worker can begin processing the next reference frame since the two reference frames resident in the shared memory will be altered if the worker is allowed to process the next reference frame. Any change in the reference frames will affect the processing of the non-reference frames. To improve the efficiency, when workers are processing non-reference frames (B frames), the reference frame immediately following them is allowed to be processed, but the results are not updated in the shared memory. The results are buffered temporarily and the shared memory is updated during synchronization. To minimize the synchronization overhead, intermediate results are temporarily buffered and the final results are computed during synchronization after every I frame in the MPEG-1 stream. As before, the inherent dependencies between I, P and B frames limit the extent of parallelism. Only the workers handling slices from B frames and the immediately succeeding I or P frame can execute in parallel. There are at least two circular task queues used in this implementation. The master process uses one queue to fill all slices belonging to a particular frame and proceeds to fill the other queue for a new frame. This allows the insert-into-queue operation performed by the master process and the remove-from-queue operations performed by the worker processes to be overlapped without entailing a high synchronization overhead.

3.3.1. Parsing using DC images and motion vectors (Approach 1)

The master process scans the MPEG-1 stream for the start of a new slice and inserts the slice along with the appropriate GOP, frame and slice headers into the

```
procedure Main Process - Slice
begin
    while Not end of MPEG stream
    do
        Scan stream for next slice start code
        Read frame
        if ( Task Queue == Full )
        then
            Wait
        Insert slice and corresponding frame header, GOP header in task queue
        Signal workers
    end
end
```

Fig. 16. Pseudocode outline of the master process for parallel video parsing at the slice level (Approach 1).

task queue. Processing takes place in the order of the frame numbers. However, the slices within a frame are processed in any order. The frames that appear early in the MPEG stream have to be processed before the later frames. So all the workers having slices from the lowest numbered frame take precedence. If the frame is a B frame, the precondition of the availability of both its reference frames must be met. If it is a reference frame, no other frames except the preceding B frames should be active. The pseudocode outline of the master process is shown in Fig. 16.

The worker process for parallel video parsing at the slice level is very similar to its frame-level counterpart. The worker processes compute all the intermediate results and synchronize at the end of every frame. In contrast to the frame-level implementation where a frame is processed by a single worker, here a frame is processed by all workers. The master and worker processes are very similar in both cases, whether the DC values or the motion vectors are used to determine scene changes. The pseudocode outline of the worker process is shown in Fig. 17.

3.3.2. Parsing using DC image strips (Approach 2)

The master process in Approach 2 is identical to the one in Approach 1 (Fig. 16). However the structure of the worker process in Approach 2 more closely resembles that of the worker process in Approach 2 of the frame-level implementation. The three strips, horizontal, vertical and diagonal are extracted from the three DC images of each frame. Since the workers work with slices rather than entire frames, the intermediate data from all the slices are used to build a shared frame. The last worker to finish in a frame extracts the 3 strips from the shared frame that has been built. At the end of each frame, the differences are computed and the current values are stored in the shared memory for the next frame. Dissolves are implemented using the same k difference approach that is used in Approach 1 with the DC images. The worker that processes the last slice within a frame, computes the sum of the strips and computes the difference if the value of the kth frame exists locally. The difference is then thresholded to check for peaks and plateaus. A more parallel version could utilize all workers in the computation of the sums and differences, but is highly

```
procedure Worker Process - Approach 1, Slice-based
begin
    while Not end of stream
    do
        if ( Task Queue == Empty )
        then
            Wait
        Read slice
        if ( Last frame == I frame )
        then
            PreProcess
        while ( My frame NOT lowest )
        do
            Wait
        end
        if ( My frame == I frame )
        then
            Signal all workers
            PreProcess
        Extract part of DC images from the slice
        Store motion vector information in shared memory
        if ( Current Frame == I/P frame )
        then
            Barrier synchronization : Serial execution starts here
            Update shared data and reference frames
            Serial execution ends here
        Signal other workers and master process
    end
end


procedure PreProcess
begin
    Barrier Synchronization
    for all stored frames
    do
        Compute the DC differences between frames i and i − 1
        Compute motion distances between frames i and i − 1
    end
    Barrier Synchronization
end
```

Fig. 17. Pseudocode outline of the worker process for parallel video parsing at the slice level (Approach 1).

complex. The pseudocode outline for the worker process for Approach 2 is given in Fig. 18.

```
procedure Worker Process - Approach 2, Slice-based
begin
    while Not end of stream
    do
        if ( Task Queue == Empty )
        then
            Wait
        Read slice
        while ( My frame NOT lowest )
        do
            Wait
        end
        Extract the horizontal, vertical and diagonal strips from DC images
        Barrier synchronization : Serial execution starts here
        Find edge differences between strips i and i − 1
        Store the sum of DC values of all strips in shared memory
        if ( kth sum exists )
        then
            Compute the kth difference between frames i and i + k
        if ( (i − k)th sum not exists )
        then
            Compute the kth difference between frames i − k and i
        Serial execution ends here
        if ( Current Frame == I/P frame )
        then
            Barrier synchronization : Serial execution starts here
            Update shared data and reference frames
            Serial execution ends here
        Signal workers and master process
    end
end
```

Fig. 18. Pseudocode outline of the worker process for parallel video parsing at the slice level (Approach 2).

4. Analytical evaluation of performance

The performance of the various parallel MPEG-1 video parsing schemes discussed in the previous section were analyzed in terms of speedup, synchronization overhead and general memory requirements. The following terminology is introduced before the analytical expressions for speedup and synchronization overhead are derived.

T_c: Time taken by a worker to process a single task,
T_c^1: Computation time for a 1-worker system. This includes time taken by the worker to access the task queue,

$T^1_{w,m}$: Wait time experienced by the worker in a 1-worker system waiting for the master to initialize the task queue,
$T^n_{w,m}$: Total wait time experienced by all the workers in an n-worker system waiting for the master to initialize the task queue,
$T^n_{w,w}$: Total time spent in synchronization between the workers in an n-worker system,
$T_{w,r}$: Time taken by a worker to remove a job from the queue,
$T_{m,i}$: Time taken by the master to construct and insert a job into the queue,
T^n_c: Computation time for an n-worker system,
T^1: Total time for a 1-worker system,
T^n: Total time for an n-worker system,
S_t: Size of the task (GOP, slice or frame),
S_q: Size of the task queue in terms of number of tasks,
n_t: Total number of tasks,
n: Total number of worker processes.

4.1. Parallel parsing at the GOP level

The total time taken by a 1-processor system T^1 can be expressed as:

$$T^1 = T^1_{w,m} + T^1_c \tag{1}$$

where

$$T^1_c = n_t T_c + n_t T_{w,r} = n_t K_2 S_t + n_t K_3 S_t = n_t (K_2 + K_3) S_t \tag{2}$$

and K_2 and K_3 are constants. The computation time T_c for a task and the time taken to remove a task from the queue $T_{w,r}$ are proportional to the task size S_t. The wait time at the task queue for 1-processor system $T^1_{w,m}$ can be expressed as

$$T^1_{w,m} = T^1_{(w,m)_i} + T^1_{(w,m)_l} \tag{3}$$

where $T^1_{(w,m)_i}$ is the initial wait time encountered by the worker processes while the task queue is being filled by the master process and $T^1_{(w,m)_l}$ is the wait time encountered by the worker process at the task queue at any other time during execution. It is clear that

$$T^1_{(w,m)_i} = K_1 S_t = T_{m,i} \tag{4}$$

where K_1 is a constant, i.e., the initial wait time $T^1_{(w,m)_i}$, which includes the time taken by the master process to construct a task and insert it in the task queue, is directly proportional to the task size. If the execution of the worker process is overlapped with the time taken by the master process to construct and insert tasks in the task queue and if the time taken by a worker process on a task is greater than the time taken by the master process to construct and insert a task in the task queue, i.e., $T^1_c > T_{m,i}$ then $T^1_{(w,m)_l} = 0$. That is to say, there is a new task already present in the task queue when the worker process has completed its current task. Hence

$$T^1_{\mathrm{w,m}} = T^1_{(\mathrm{w,m})_\mathrm{i}} = K_1 S_\mathrm{t} \tag{5}$$

Note that the task queue is assumed to be of size 1 since a larger queue size is not needed for a 1-worker system. From Eqs. (1)–(5)

$$T^1 = K_1 S_\mathrm{t} + (K_2 + K_3) n_\mathrm{t} S_\mathrm{t} \tag{6}$$

For an n-worker system, the initial wait time at the task queue $T^n_{(\mathrm{w,m})_\mathrm{i}}$ is given by $T^n_{(\mathrm{w,m})_\mathrm{i}} = K_1 S_\mathrm{q} S_\mathrm{t}$. Recall that the time T_c taken by a single worker process to process a single task is given by $T_\mathrm{c} = K_2 S_\mathrm{t}$. It can be seen that if $T_\mathrm{c} > T^n_{(\mathrm{w,m})_\mathrm{i}} + (n-1)T_{\mathrm{w,r}}$, the master process will have reinitialized the task queue before a worker process needs to access the task queue for its next task. This implies that the wait time for subsequent access to the task queue $T^n_{(\mathrm{w,m})_\mathrm{l}} = 0$, i.e., no worker process will be kept waiting for subsequent access to the task queue. The synchronization overhead between the worker processes $T^n_{\mathrm{w,w}}$ is given by

$$T^n_{\mathrm{w,w}} = T^n_{(\mathrm{w,w})_\mathrm{i}} + T^n_{(\mathrm{w,w})_\mathrm{l}} \tag{7}$$

$T^n_{(\mathrm{w,w})_\mathrm{i}}$ is the initial wait time at the task queue where a worker needs to wait while another worker is accessing the task queue and $T_{(\mathrm{w,w})_\mathrm{l}}$ is the time spent by a worker in synchronizing with other workers over the duration of the computation. For a GOP-level implementation, since each worker processes its GOP independently of the other workers, $T_{(\mathrm{w,w})_\mathrm{l}} = 0$ whereas $T^n_{(\mathrm{w,w})_\mathrm{i}} = (n-1)K_3 S_\mathrm{t}$. The time T^n_c spent in computation by an n-worker system on n processors can be expressed as:

$$T^n_\mathrm{c} = \frac{T^1_\mathrm{c}}{n} = (K_2 + K_3)\left(\frac{n_\mathrm{t}}{n}\right) S_\mathrm{t} \tag{8}$$

In Eq. (8) it is assumed that each worker process is assigned to an independent processor within the SMP. Thus

$$T^n = T^n_\mathrm{c} + T^n_{\mathrm{w,m}} + T^n_{\mathrm{w,w}} = (K_2 + K_3)\left(\frac{n_\mathrm{t}}{n}\right) S_\mathrm{t} + K_1 S_\mathrm{q} S_\mathrm{t} + (n-1)K_3 S_\mathrm{t} \tag{9}$$

The speedup $\sigma(n, n_\mathrm{t})$ is given by

$$\begin{aligned}\sigma(n, n_\mathrm{t}) &= \frac{T^1}{T^n} = \frac{K_1 S_\mathrm{t} + (K_2 + K_3) n_\mathrm{t} S_\mathrm{t}}{K_1 S_\mathrm{q} S_\mathrm{t} + \frac{(K_2+K_3) n_\mathrm{t} S_\mathrm{t}}{n} + (n-1)K_3 S_\mathrm{t}} \\ &= \frac{n\left(1 + \frac{K_1}{(K_2+K_3) n_\mathrm{t}}\right)}{\left(1 + S_\mathrm{q} \frac{K_1}{K_2+K_3} \frac{n}{n_\mathrm{t}} + \frac{n(n-1)}{n_\mathrm{t}} \frac{K_3}{K_2+K_3}\right)}\end{aligned} \tag{10}$$

Note that for most video streams, $K_1, K_3 \ll K_2$ and $S_\mathrm{q} \ll n_\mathrm{t}$, i.e., the time taken to construct a task, queue a task or remove a task from the queue is much smaller than the time taken to process it, and the size of the task queue is much smaller than the number of tasks (GOPs) in the video stream. In the limit as $n_\mathrm{t} \to \infty$, i.e., as the video stream becomes longer it can be seen from Eq. (10) that

$$\sigma_1(n) = \lim_{n_t \to \infty} \sigma(n, n_t) = n \tag{11}$$

Also, for a given value of n_t, in the limit as $n \to \infty$, i.e., as the number of processors are increased it can be seen from Eq. (10) that

$$\sigma_k(n_t) = \lim_{n \to \infty} \sigma(n, n_t) = 0 \tag{12}$$

Thus, better speedup is realized for video streams that are longer (i.e., have a larger number of GOPs) in comparison to the number of processors in the SMP i.e., the ratio $\frac{n_t}{n}$ is very high. The synchronization overhead $\omega(n, n_t)$ is given by:

$$\begin{aligned}\omega(n, n_t) &= \frac{T^n_{w,m} + T^n_{w,w}}{T^n} = \frac{K_1 S_q S_t + (n-1) K_3 S_t}{K_1 S_q S_t + (n-1) K_3 S_t + \frac{(K_2+K_3) n_t S_t}{n}} \\ &= \frac{1}{1 + \left(\frac{K_2+K_3}{K_1 S_q + (n-1) K_3}\right)\left(\frac{n_t}{n}\right)}\end{aligned} \tag{13}$$

In the limit as $n_t \to \infty$ it can be seen that

$$\omega_1(n) = \lim_{n_t \to \infty} \omega(n, n_t) = 0 \tag{14}$$

and

$$\omega_k(n_t) = \lim_{n \to \infty} \omega(n, n_t) = 1 \tag{15}$$

These results are to be expected since the initial wait time at the task queue becomes negligible compared to the total processing time as the length of the video stream increases for a given value of number of processors n. Also, for a given video stream, the initial wait time at the task queue becomes increasingly significant as the number of processors n increases.

4.2. Parallel parsing at the frame level

As in the case of parallel parsing at the GOP level,

$$T^1 = T^1_c + T^1_{w,m} \tag{16}$$

and

$$T^n = T^n_c + T^n_{w,m} + T^n_{w,w} \tag{17}$$

It can be seen that

$$T^1_c = (K_2 + K_3) n_t S_t \tag{18}$$

and

$$T^1_{w,m} = T^1_{(w,m)_i} + T^1_{(w,m)_l} = K_1 S_t + 0 = K_1 S_t \tag{19}$$

Hence

$$T^1 = K_1 S_t + (K_2 + K_3) n_t S_t \tag{20}$$

Also the time T_c taken by a worker to process a task can be expressed as

$$T_c = T_{c,1} + T_{c,2} \tag{21}$$

where $T_{c,1}$ represents the inherently serial portion and $T_{c,2}$ the parallelizable portion of T_c. Let $T_{c,1} = K_4 S_t$ and $T_{c,2} = K_5 S_t$ such that $K_4 + K_5 = K_2$. Hence

$$T_c^n = K_4 n_t S_t + \frac{K_5 n_t S_t}{\alpha n} \tag{22}$$

where $0 < \alpha < 1$ takes into account the synchronization and communication overhead between the n worker processes i.e. the term $T^n_{(w,w)_l}$ in

$$T^n_{w,w} = T^n_{(w,w)_i} + T^n_{(w,w)_l} \tag{23}$$

where $T^n_{(w,w)_i} = (n-1)K_3 S_t$ as in the GOP case. Hence

$$\begin{aligned} T^n &= T^n_{w,m} + T^n_{w,w} + T^n_c = T^n_{(w,m)_i} + T^n_{(w,m)_l} + T^n_{(w,w)_i} + T^n_{(w,w)_l} + T^n_c \\ &= K_1 S_q S_t + (n-1)K_3 S_t + K_4 n_t S_t + \frac{K_5 n_t S_t}{\alpha n} \end{aligned} \tag{24}$$

Note that $T^n_{(w,m)_l} = 0$ as in the GOP case. Hence

$$\begin{aligned} \sigma(n, n_t) = \frac{T^1}{T^n} &= \frac{K_1 S_t + (K_2 + K_3) n_t S_t}{K_1 S_q S_t + (n-1)K_3 S_t + K_4 n_t S_t + \frac{K_5 n_t S_t}{\alpha n}} \\ &= \frac{K_1 + (K_2 + K_3) n_t}{K_1 S_q + (n-1)K_3 + K_4 n_t + \frac{K_5 n_t}{\alpha n}} \\ &= \frac{\alpha n \left(1 + \frac{K_1}{n_t (K_3 + K_4 + K_5)}\right)}{\alpha \left(\frac{n}{n_t}\right) S_q \left(\frac{K_1}{K_3 + K_4 + K_5}\right) + \frac{\alpha n (n-1)}{n_t} \left(\frac{K_3}{K_3 + K_4 + K_5}\right) + \alpha n \left(\frac{K_4}{K_3 + K_4 + K_5}\right) + \frac{K_5}{K_3 + K_4 + K_5}} \end{aligned} \tag{25}$$

Note that in the frame-level implementation, the parameter α is related to the fraction of B frames in the GOP. Recall that the presence of I and P frames imposes sequential execution in parallel video parsing at the frame level. The degree of parallelism that can be exploited at the frame level increases with the fraction of B frames in the GOP. Also note that the inclusion of the factor α in T^n_c takes into account $T^n_{(w,w)_l}$ and a separate expression for $T^n_{(w,w)_l}$ is not necessary. In the limit as $n_t \to \infty$, i.e., as the number of frames in a GOP becomes increasingly large,

$$\sigma_1(n) = \lim_{n_t \to \infty} \sigma(n, n_t) = \frac{\alpha n}{\left(\frac{K_5}{K_3 + K_4 + K_5} + \alpha n \frac{K_4}{K_3 + K_4 + K_5}\right)} \tag{26}$$

Furthermore, when the percentage of parallelizable code is high, $\frac{K_4}{K_2} = \frac{K_4}{K_4 + K_5} \to 0$ and hence $\alpha n \frac{K_4}{K_3 + K_4 + K_5} \to 0$. In which case,

$$\sigma_1(n) \approx \alpha \left(1 + \frac{K_3}{K_2}\right) n \tag{27}$$

which is linear in n. As the number of processors increase, i.e., $n \to \infty$,

$$\lim_{n\to\infty} \sigma_1(n) = \lim_{n\to\infty} \frac{1}{\frac{K_4}{K_3+K_4+K_5} + \frac{K_5}{\alpha n(K_3+K_4+K_5)}} = \frac{K_3+K_4+K_5}{K_4} = \frac{K_3+K_2}{K_4} \tag{28}$$

Thus the speedup reaches an asymptotic limit as the number of processors is increased for large GOP sizes causing the efficiency to fall to zero.

The synchronization overhead ω is given by

$$\omega(n, n_t) = \frac{T^n_{w,m} + T^n_{w,w}}{T^n} \tag{29}$$

The value of $T^n_{w,w}$ is given by $T^n_{w,w} = T^n_{(w,w)_i} + T^n_{(w,w)_l}$ where $T^n_{(w,w)_i} = (n-1)K_3S_t$ as in the GOP case and

$$T^n_{(w,w)_l} = \frac{T_{c,2}}{\alpha n} - \frac{T_{c,2}}{n} = K_5 S_t \left(\frac{n_t}{n}\right)\left(\frac{1}{\alpha} - 1\right) \tag{30}$$

Hence

$$\begin{aligned}\omega(n, n_t) &= \frac{K_1S_qS_t + (n-1)K_3S_t + K_5S_t\left(\frac{n_t}{n}\right)\left(\frac{1}{\alpha}-1\right)}{K_1S_qS_t + (n-1)K_3S_t + n_tK_4S_t + \frac{n_t}{\alpha n}K_5S_t} \\ &= \frac{\frac{K_1S_q+(n-1)K_3}{n_t} + \frac{K_5}{n}\left(\frac{1}{\alpha}-1\right)}{\frac{K_1S_q+(n-1)K_3}{n_t} + \left(K_4 + \frac{K_5}{\alpha n}\right)}\end{aligned} \tag{31}$$

From Eq. (31) in the limit as $n_t \to \infty$,

$$\lim_{n_t\to\infty} \omega(n, n_t) = \omega_l(n) = \frac{\frac{K_5}{n}\left(\frac{1}{\alpha}-1\right)}{K_4 + \frac{K_5}{\alpha n}} = \frac{1-\alpha}{\left(1 + \alpha\frac{K_4}{K_5}n\right)} \tag{32}$$

In the limit as $\frac{K_4}{K_5} \to 0$, i.e., as the parallelizable fraction of the code increases, from Eq. (32), it can be seen that

$$\lim_{\frac{K_4}{K_5}\to 0} \omega_l(n) = 1 - \alpha \tag{33}$$

4.3. Parallel parsing at the slice level

The analysis of parallel video parsing at the slice level is along the same lines as the analysis at the frame level. The only difference is that here n_t is the average number of slices in a frame. The variable α is interpreted as the fraction of worker processes that can proceed concurrently. In fact, in most cases $\alpha \approx 1$. The only case where the value of α deviates substantially from 1 is when the worker processes processing a subsequent reference frame are finished whereas the worker processes processing a previous B frame have not yet finished processing. This is an infrequent occurrence since the processing of frames is typically completed in the temporal sequence of their occurrence in the video stream.

5. Experimental evaluation of performance

The performance of the various parallel MPEG-1 video parsing schemes discussed in the previous section was experimentally verified in terms of speedup, synchronization overhead and general memory requirements. All our experiments were conducted on a SUN Enterprise 10 000 server which is a shared memory symmetric multiprocessor (SMP) machine with 32 Ultrasparc 400 MHz processors and 32 GB of shared memory running the Solaris 8 operating system. All programs are compiled with gcc version 2.7.2.2. The programs use the Solaris shared memory library. All the processes in the system are traditional UNIX processes. Shared memory is used as the medium of communication. Shared memory is preferred over message queues since the latter have strict restrictions on the sizes of allowed messages.

5.1. Parallel parsing at the GOP level

5.1.1. Approach 1

Speedup curves for the parallel MPEG-1 parsing at the GOP-level with Approach 1 using DC images are shown in Fig. 19(a) and (b). The speedup is computed with

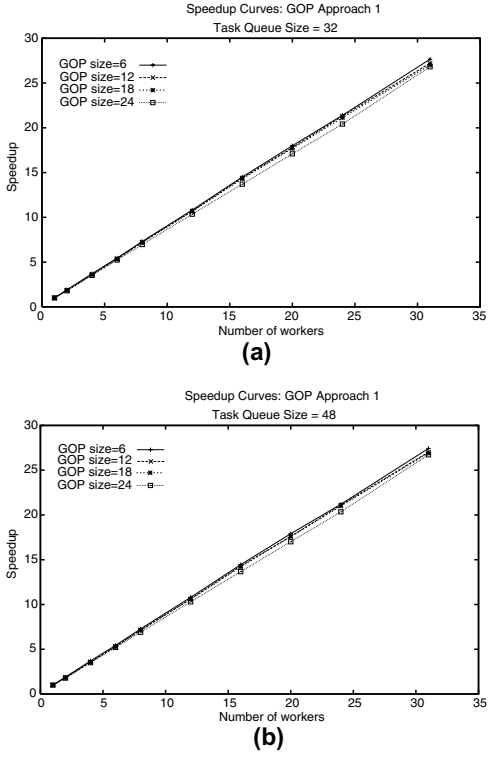

Fig. 19. Speedup curves for parallel MPEG-1 parsing at the GOP level with Approach 1 using DC images. (a) Task queue size = 32 and (b) Task queue size = 48.

respect to the time taken by a single worker system. It can be seen that the speedup increases almost linearly with the increase in number of workers up to 31. Since the maximum number of processors in the SMP is 32 and one of the processors is assigned to the master process, the maximum number of workers possible (with an assignment of one processor per worker) is 31. If more than 31 worker processes are executed, the processes are time-shared on the processors available on the SMP. Two different sizes for the task queue (32 and 48) with upto 31 worker processes were examined. As the graphs in Fig. 19(a) and (b) show, there is not much difference in the speedup values. It should be noted that the time taken to insert one GOP into the queue is much less than the time taken to process a GOP. Since there are 31 workers, for task queue sizes greater than 32, the workers do not spend any time waiting for the task queue to be filled. Hence speedup does not vary for task queue sizes greater than 32. Since the GOP level utilizes the coarsest granularity of parallelism with minimal synchronization, the speedup is observed to scale in a manner that is almost linear. This is in conformity with the theoretical analysis presented in the previous section.

The speedup in Fig. 19(a) and (b) was seen to be affected by the GOP size. The GOP size refers to the number of frames in the traditional GOP from which the custom GOPs are constructed. Four different GOP sizes were examined. The GOP size was found to directly impact the load imbalance among the worker processes and hence the speedup. Since all the worker processes execute on processors of the same speed, if the GOPs are not equally divided among the workers, load imbalance would result. The effect of this imbalance is more noticeable when the size of the GOP is large and the number of GOPs in the MPEG stream is small. Also, the

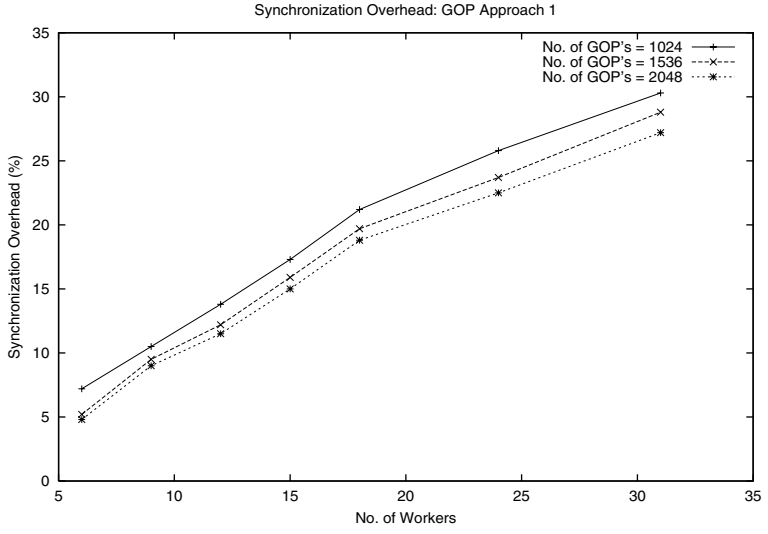

Fig. 20. Synchronization overhead for parallel MPEG-1 parsing at the GOP level with Approach 1 using DC images.

disparity in the speedup values, for GOPs of different sizes, is observed to increase with the number of worker processes. This can be attributed to fact that as the number of worker processes increases, so does the initial wait time.

Synchronization overhead is measured as the percentage of the total execution time that is spent in synchronization primitives. Fig. 20 shows the synchronization overhead as a function of the number of worker processes for Approach 1. Three different values for the total number of tasks (i.e., GOPs) n_t are considered, 1024, 1536 and 2048. The synchronization waits are due to stalls resulting from access to shared data (which includes the task queue) and stalls resulting from data dependencies. The initial wait is for the task queue to get filled for the first time. The initial wait time increases with an increase in the number of worker processes. The later wait times, caused by inter-process synchronization, are small since the worker processes work independently of each other. The synchronization overhead is thus observed to increase with an increase in the number of worker processes. The synchronization overhead is also observed to reduce as the number of GOPs in the video stream (i.e., tasks) increases for a given number of worker processes. These observations are in conformity with the theoretical analysis presented in the previous section.

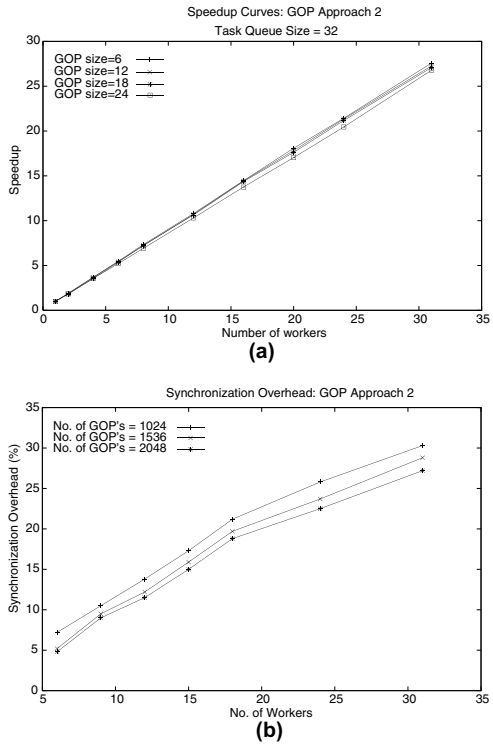

Fig. 21. Parallel MPEG-1 parsing at the GOP level with Approach 2 using DC image slices. (a) Speedup curves and (b) synchronization overhead curves.

5.1.2. Approach 2

Speedup curves for the parallel MPEG-1 parsing at the GOP-level with Approach 2 using DC image strips are shown in Fig. 21(a). Four different sizes of GOPs are used to test the speedup. As in the case of Approach 1, the smaller-size GOPs perform better than the larger-size GOPs. From Fig. 21(b) it can be observed that the synchronization overhead increases with an increase in the number of worker processes. This is due to the increased initial wait experienced by the worker processes while the master process fills the task queue. As in the case of Approach 1, the synchronization overhead is also observed to reduce as the number of GOPs in the video stream (i.e., tasks) increases for a given number of worker processes.

5.2. Parallel parsing at the frame level

5.2.1. Approach 1

Fig. 22(a) and (b) shows the speedup curves for parallel video parsing at the frame level with Approach 1 which uses DC images. Two different sizes for the task queue were examined, i.e., 32 frames and 48 frames. The speedup was observed to be sub-linear because of high synchronization overhead which limits parallelism. The speedup was observed to level off as the number of worker processes exceeded 12 (Fig.

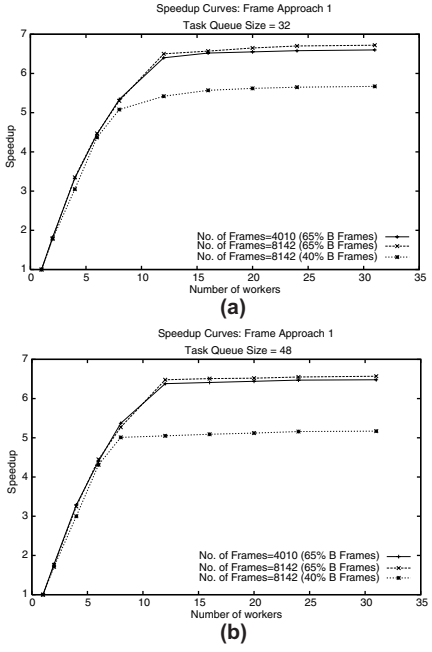

Fig. 22. Speedup curves for parallel MPEG-1 parsing at the frame level with Approach 1 using DC images. (a) Task queue size = 32 and (b) task queue size = 48.

22(a) and (b)). The size of the task queue did not make a noticeable difference. Load imbalance was not a significant factor because of the finer granularity of task and data decomposition. The speedup values were seen to be particularly sensitive to the fraction of B frames in the video stream. Video streams with a higher percentage of B frames were seen to yield higher speedup values (Fig. 22). For a given percentage of B frames, the speedup values were observed to be higher for video streams with more frames. Overall, the speedup values were observed to be more sensitive to the percentage of B frames in the video stream (which determines the value of α) than the total number of frames in the video stream (which determines the value of n_t).

The frame-level implementation involves high synchronization overhead. The overhead, measured as the percentage of total time spent in synchronization, was observed to be almost constant with respect to an increasing number of worker processes. The initial wait time, as discussed in the analysis of parallel video parsing at the GOP-level, was present here as well but was overshadowed by the enormous amount of subsequent synchronization. The synchronization overhead did not depend on the number of frames in the MPEG-1 stream because every frame is associated with the same amount of processing and synchronization overhead, rather it was seen to depend on the percentage of B frames in the video stream. As shown in Fig. 23, for a video stream with 8142 frames, the synchronization overhead was approximately 55% with 65% B frames and approximately 65% with 40% B frames. For a video stream with 4010 frames and 65% B frames, the synchronization overhead was observed to be approximately 59%. Thus, for a given number of processors and percentage of B frames, the synchronization overhead was seen to decrease with an

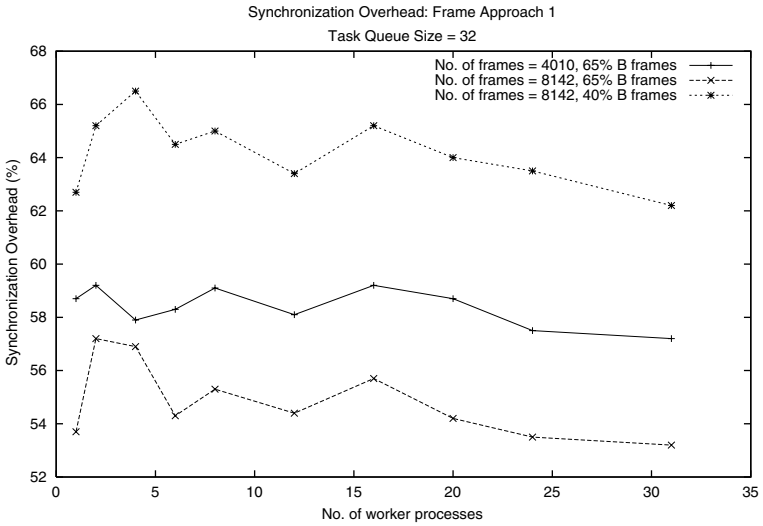

Fig. 23. Synchronization overhead for parallel MPEG-1 parsing at the frame level with Approach 1 using DC images.

increasing number of frames in the video stream. However, for a a given number of processors and number of frames in the video stream, the synchronization overhead was observed to be far more sensitive to the percentage of B frames in the video stream; the synchronization overhead increased sharply even with a small decrease in the percentage of B frames in the video stream.

5.2.2. Approach 2

Fig. 24(a) depicts the speedup for parallel video parsing at the frame level using DC image slices (Approach 2). As in the case of Approach 1, the speedup was seen to be sublinear due to the high synchronization overhead which limits the extent of parallelism. The speedup was observed to level off for more than 12 worker processes. Approach 2 was also observed to incur high synchronization overhead as depicted in Fig. 24(b). The synchronization overhead was observed to be almost constant with respect to the number of worker processes. As in the case of Approach 1, the synchronization overhead was observed to be more sensitive to the percentage of B frames in the video sequence rather than the total number of frames in the video sequence.

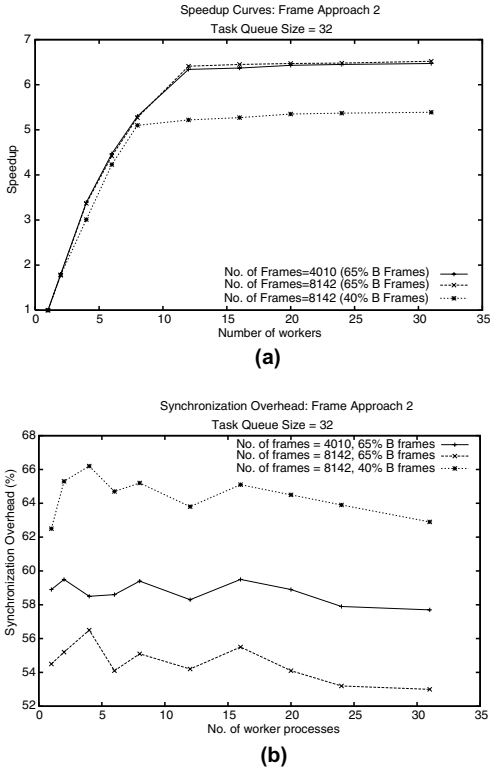

Fig. 24. Parallel MPEG-1 parsing at the frame level with Approach 2 using DC image slices. (a) Speedup curves and (b) synchronization overhead curves.

5.3. Parallel parsing at the slice level

5.3.1. Approach 1

Experiments were conducted for four different frame resolutions. The resolution affects the number of slices in a frame. Four different frame resolutions were considered: 32 slices/frame, 64 slices/frame, 96 slices/frame and 128 slices/frame. Also, two different sizes for the task queue (32 and 48) were considered. The size of a single slice, however, was kept constant. As Fig. 25(a) and (b) shows, the slice-level implementation yielded speedup figures that are better than the frame-level implementation but worse than the GOP-level implementation. Since the slice-level implementation is similar to its frame-level counterpart, it is constrained by a similar synchronization overhead. The improvement over the frame-level implementation can be attributed to the parallel processing of a frame by several worker processes. In the frame-level implementation, a worker works on a single frame and if that frame happens to be a reference frame, then that worker process is the only one that is permitted to run whereas the other workers in the system are forced to remain idle.

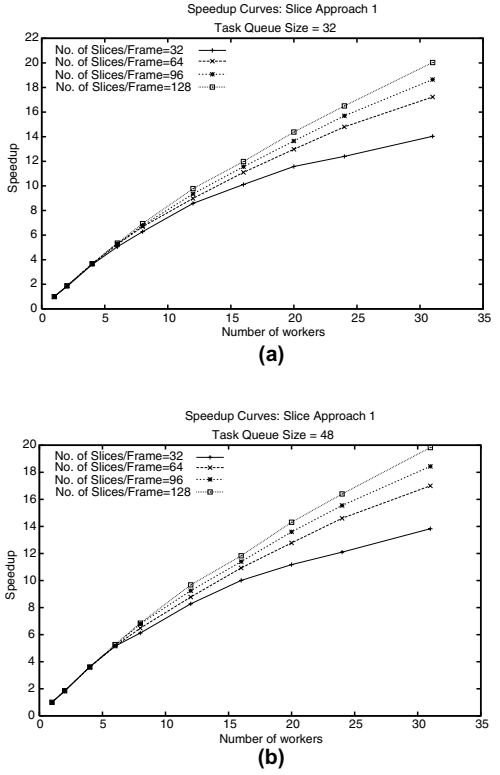

Fig. 25. Speedup curves for parallel MPEG-1 parsing at the slice level with Approach 1 using DC images. (a) Task queue size = 32 and (b) task queue size = 48.

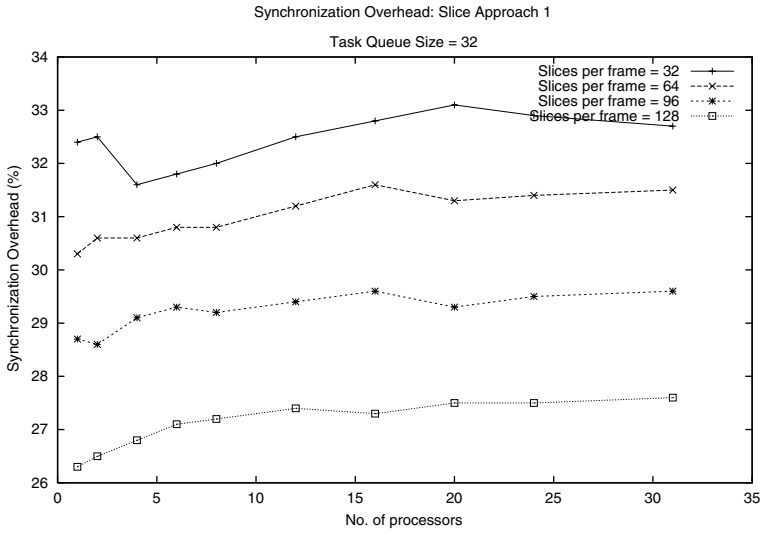

Fig. 26. Synchronization overhead for parallel MPEG-1 parsing at the slice level with Approach 1 using DC images.

In the slice-level implementation, almost all the workers are active during the processing of all frames, thus improving the speedup.

Fig. 25(a) and (b) shows the speedup results for four different frame resolutions. The speedup performance was seen to be better for larger frame resolutions due to the relatively infrequent synchronization amongst the workers which is in conformity with the theoretical analysis. Load imbalance was not seen to be a major issue because of the fine granularity of task and data decomposition. Although the speedup was sublinear in the number of worker processes, the speedup values were better than those in the case of the frame-level implementation. Also, the speedup curve in the case of the slice-level implementation did not display the saturation (i.e., leveling off) effect that it did in the case of the frame-level implementation.

Fig. 26 depicts the synchronization overhead for the slice-level implementation. The synchronization overhead was observed to decrease as the number of slices in a frame (i.e., frame resolution) was increased. This was expected since the workers synchronize less frequently and spend more time in computation between synchronization when the frame resolution is increased. The synchronization overhead also exhibited a slight increasing trend with an increase in the number of worker processes. These experimental observations were in conformity with the theoretical analysis presented earlier.

5.3.2. Approach 2

The experimental results in the case of Approach 2, were similar to those in the case of Approach 1. As shown in Fig. 27(a), the speedup was observed to be sublinear in the number of worker processes. However the speedup values were higher than

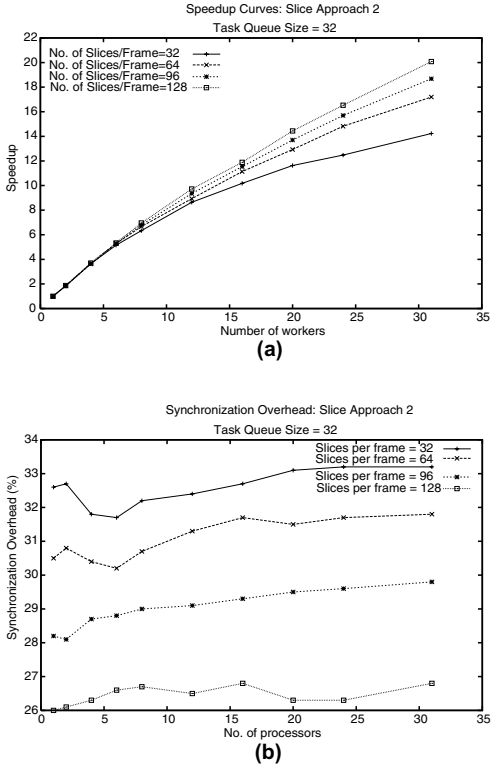

Fig. 27. Parallel MPEG-1 parsing at the slice level with Approach 2 using DC image slices. (a) Speedup curves and (b) synchronization overhead curves.

those in the corresponding frame-level implementation. As in the case of Approach 1, the speedup values were better for higher frame resolutions. The synchronization overhead is depicted in Fig. 27(b). As in the case of Approach 1, the synchronization overhead was seen to decrease with an increase in the frame resolution. For a given frame resolution, the synchronization overhead was seen to exhibit a slight increasing trend with an increase in the number of worker processes.

5.4. Memory requirements

The system-wide memory requirements are depicted in Fig. 28 for the GOP-level, frame-level and slice-level implementations. The GOP-level implementation was seen to consume the most memory and the memory requirement increased with an increase in the size of the GOP. The memory requirements also increased with an increase in size of the task queue since for larger task queues, a larger number of GOPs were stored in memory. In the GOP-level implementation, each worker process had to retain 3 frames and the associated data in its local memory. As the number of worker processes was increased, so did this local memory requirement.

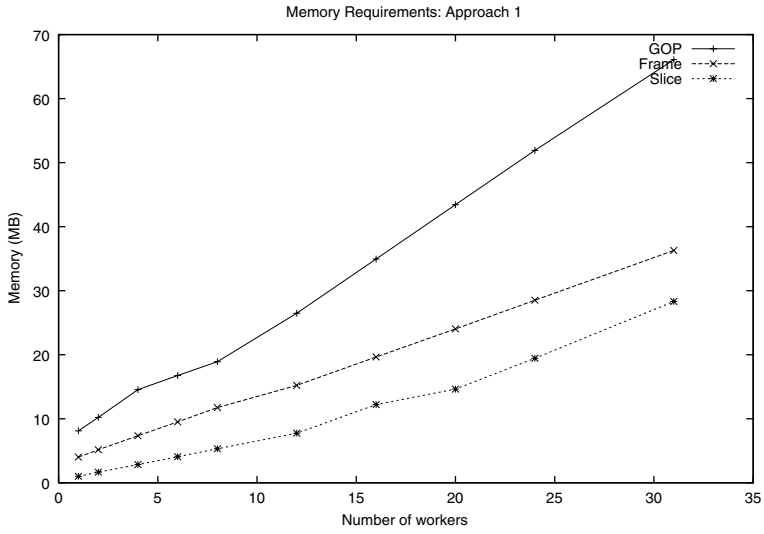

Fig. 28. Memory requirements for the three levels of parallel processing granularity: GOP, frame and slice.

For the frame-level implementation, the memory requirements were seen to be less stringent. The principal contender for memory was the task queue. However, since each slot in the task queue holds a single frame it occupied much less space than the slot in the task queue in the GOP-level implementation. The 3 frames resident in memory and their associated data were common to all the worker processes. This memory requirement did not increase with the number of worker processes which is in contrast to the GOP-level implementation. The slice-level implementation boasted the least memory requirement. The task queue occupied the least amount of memory since each slot in the task queue holds only a single slice. As in the case of the frame-level implementation, the 3 frames resident in memory and their associated data were common to all the workers and thus this memory requirement did not increase with the number of worker processes.

5.5. Multiple task queues

The video parsing system discussed thus far used a single global task queue that was shared by all the worker processes. The slice-level implementation used two or more queues which were also shared by all the workers. Since a shared global queue results in contention among the processors, an improved version of the system was implemented in which each worker was provided with its own task queue. The rest of the system and its working were unchanged.

In the improved version of the video parsing system, the master process scans the MPEG-1 stream and queues the tasks (custom GOPs, frames or slices) onto each of the worker queues in a round-robin fashion. The worker processes read the tasks

from their respective task queues. For the slice method, each worker has its individual set of multiple queues. The remainder of the processing by the workers is the same as that in the case of the single task queue implementation. It should be noted that the master process would still need to synchronize with the worker processes but the need for synchronization amongst the worker processes is reduced. The pseudocode description of the master process for the multiple-queue version of the parallel video parsing algorithm at the GOP level, frame level and the slice level is given in Figs. 29–31, respectively. The pseudocode description of the worker process for the multiple-queue implementation at the GOP level, frame level and slice level is the same as that of its single-queue counterpart.

Fig. 32 depicts the synchronization overhead for the multiple-queue implementation of the video parsing algorithm based on the GOP level of parallelism. The synchronization overhead in the case of the multiple-queue implementation was observed to be lower than its single-queue counterpart—a 35% average reduction in the synchronization overhead based on the single-queue implementation. However, there is still some initial wait time involved for the worker processes in the multiple-queue implementation since the master initializes the worker task queues in a round-robin manner. This initial wait time is lower in the case of the multiple-queue implementation and increases with an increase in the number of worker processes. Hence the overall synchronization overhead was observed to have an increasing trend with respect to the number of worker processes (Fig. 32). As in the case of

```
procedure Main Process - GOP, Multiple Queue version
begin
    while Not End of MPEG stream
    do
        Scan stream for next I frame
        Construct custom-GOP
        count = 0
        while 1
        do
            if ( Task Queue of current worker != Full )
            then
                Insert custom-GOP in task queue of current worker
                Signal worker
                break
            if ( count == Number of workers )
            then
                Wait
            Move to the next worker's queue
        end
    end
end
```

Fig. 29. Main process for parallel video parsing at the GOP level with multiple task queues.

```
procedure Main Process - Frame, Multiple Queue version
begin
    while Not End of MPEG stream
    do
        Scan stream for next frame start code
        Read frame count = 0
        while 1
        do
            if ( Task Queue of current worker != Full )
            then
                Insert frame in task queue of current worker
                Signal worker
                break
            if ( count == Number of workers )
            then
                Wait
            Move to the next worker's queue
        end
    end
end
```

Fig. 30. Main process for parallel video parsing at the frame level with multiple task queues.

```
procedure Main Process - Slice, Multiple Queue version
begin
    while Not End of MPEG stream
    do
        Scan stream for next slice start code
        Read slice
        while 1
        do
            if ( Task Queue of current worker != Full )
            then
                Insert slice in task queue of current worker
                Signal worker
                break
            if ( count == Number of workers )
            then
                Wait
            Move to the next worker's queue
        end
    end
end
```

Fig. 31. Main process for parallel video parsing at the slice level with multiple task queues.

the single-queue implementation, the synchronization overhead was also observed to have a decreasing trend with respect to the number of GOPs in the video stream.

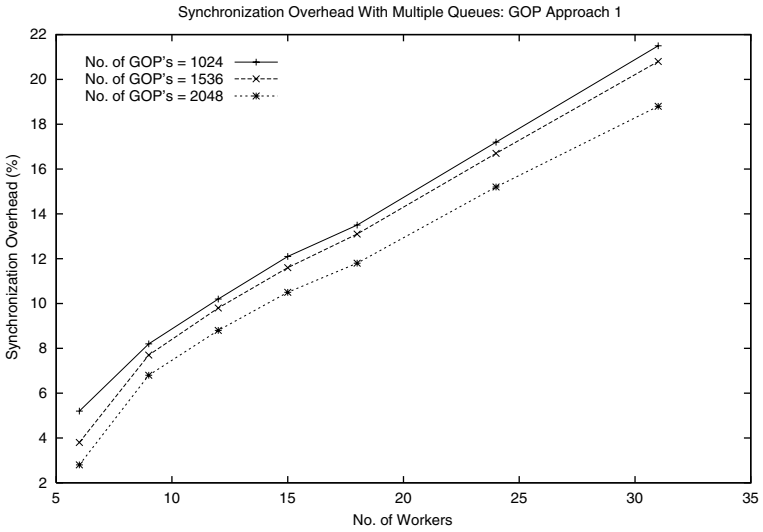

Fig. 32. Synchronization overhead for the multiple-queue GOP-level implementation: Approach 1.

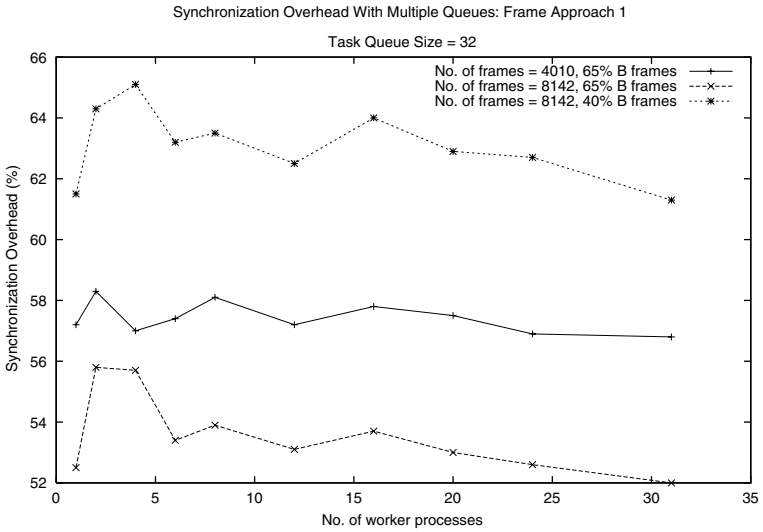

Fig. 33. Synchronization overhead for the multiple-queue frame-level implementation: Approach 1.

Fig. 33 depicts the synchronization overhead for the multiple-queue implementation of the frame-level video parsing algorithm (Approach 1). The overall synchronization overhead was seen to be lower than in the case of the single-queue implementation but only marginally—a 4% average reduction in the synchronization overhead based on the single-queue implementation. This was expected since the

initial wait time for the worker processes is a small fraction of the subsequent wait time due to data dependencies. Note that whereas the multiple-queue implementation can reduce the initial wait time for the worker processes it cannot alleviate the wait times due to stalls caused by data dependencies. As in the case of the single-queue implementation, the synchronization overhead for the multiple-queue implementation was observed to be sensitive to the fraction of B frames in the video stream i.e., the synchronization overhead increased sharply with a relatively modest decrease in the fraction of B frames in the video stream. The synchronization overhead also exhibited a decreasing trend with respect to the number of frames in the video stream in a manner similar to the single-queue implementation. The synchronization overhead was observed to be almost constant with respect to the number of worker processes in a manner similar to the single-queue implementation.

Fig. 34 depicts the synchronization overhead for the multiple-queue implementation of the slice-level video parsing algorithm (Approach 1). The overall synchronization overhead was seen to be lower than in the case of the single-queue implementation—a 12% average reduction in the synchronization overhead based on the single-queue implementation. Note that this reduction is significantly greater than the reduction observed in the case of the multiple-queue frame-level implementation but significantly lower than that observed in the case of the multiple-queue GOP-level implementation. This was expected since the initial wait time for the worker processes in the slice-level implementation as fraction of the total wait time (which includes wait times due to data dependencies) lies between the corresponding values for the frame-level implementation (very low) and the GOP-level implementation (very high). As in the case of the single-queue slice-level implementation, the synchronization overhead for the multiple-queue slice-level implementation was observed to

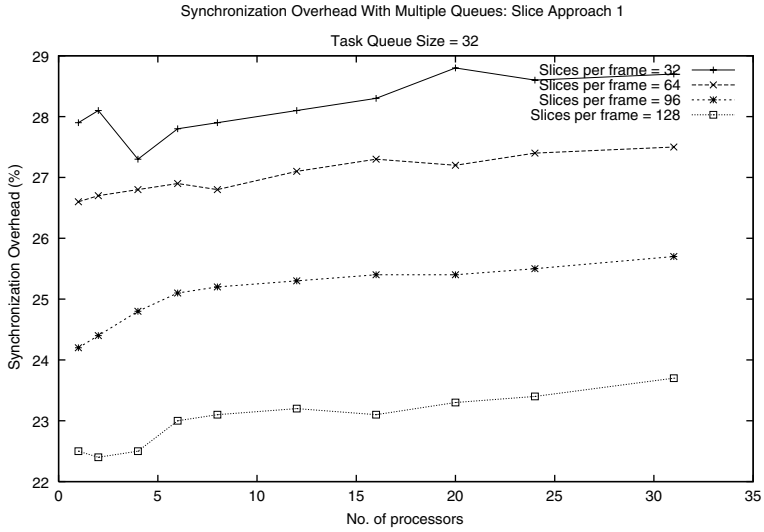

Fig. 34. Synchronization overhead for the multiple-queue slice-level implementation: Approach 1.

have a decreasing trend with respect to the frame resolution (number of slices per frame). The overall synchronization overhead was almost constant with respect to the number of worker processes as in case of the single-queue implementation.

6. Conclusions and future work

In this paper, a parallel parsing system for MPEG-1 video streams was designed and implemented, and its results were analyzed. The results showed that MPEG-1 video parsing is amenable to parallel processing on a shared-memory symmetric multiprocessor (SMP). Two different parsing algorithms were implemented and analyzed. The parallel video parsing algorithms were implemented for 3 levels of granularity of task and data decomposition, i.e., the GOP-level, frame-level and slice-level. Since MPEG-1 video exhibits inherent dependencies, the coarsest granularity of parallelism, i.e., the GOP-level, was observed to yields the best results in terms of speedup and synchronization overhead. The custom GOP-level implementation was followed by the slice-level and the frame-level implementations (in that order) in terms of speedup and synchronization overhead.

The synchronization overhead was seen to be impacted by the initial wait time when the worker task queue(s) is (are) being initialized by the master process. Since the worker processes in the GOP-level implementation process their assigned GOPs independently, the initial wait time had a greater impact on the overall synchronization overhead. In contrast, in the frame-level implementation, the worker processes had to synchronize frequently due to the inherent data dependencies between the frames within a GOP. Hence, the contribution of the initial wait time towards the overall synchronization overhead was relatively insignificant. Replacing the single task queue by multiple task queues, one for each worker process, resulted in a reduction in the initial wait time and had the most impact on the GOP-level implementation and the least on the frame-level implementation in terms of overall synchronization overhead. Overall, the speedup of the GOP-level implementation was observed to be the most scalable with respect to the number of processors within the SMP whereas the speedup of the frame-level implementation was seen to be the least scalable. The slice-level implementation was observed to lie between the GOP-level and frame-level implementations in terms of speedup, synchronization overhead and scalability of speedup and synchronization overhead with respect to the number of processors within the SMP.

Future work will include implementing the parallel video parsing algorithms on a hybrid platform consisting of a distributed memory network of nodes where each node is a shared memory symmetric multiprocessor (SMP). Parsing algorithms based on more advanced video encoding standards such as MPEG-4 and MPEG-7 will also be investigated.

References

[1] N. Adami, R. Leonardi, Identification of editing effects in image sequences by statistical modeling, in: Proceedings 1999 Picture Coding Symposium, Portland, OR, April 1999.

[2] I. Agi, R. Jagannathan, A portable fault-tolerant parallel software MPEG-1 encoder, in: Proceedings of Second IASTED/ISMM International Conference on Distributed Multimedia Systems and Applications, Stanford, CA, August 1995.
[3] G. Ahanger, T.D.C. Little, A survey of technologies for parsing and indexing digital video, Journal of Visual Communication and Image Representation 7 (1) (1996) 28–43, March.
[4] S.M. Akramullah, I. Ahmad, M.L. Liou, Parallel MPEG-2 encoder on ATM and Ethernet connected workstations. Lecture Notes in Computer Science, vol. 1557, Springer-Verlag, Berlin, 1999, pp. 572–574.
[5] F. Arman, A. Hsu, M.Y. Chiu, Image processing on compressed data for large video databases, in: Proceedings of ACM International Conference on Multimedia, Anaheim, CA, August 1993, pp. 267–272.
[6] F. Arman, R. Depommier, A. Hsu, M.Y. Chiu, Content-based browsing of video sequences, in: Proceedings of ACM International Conference on Multimedia, August 1994, pp. 97–103.
[7] S.M. Bhandarkar, A.A. Khombadia, Motion-based parsing of compressed video, in: Proceedings of IEEE International Workshop on Multimedia Database Management Systems, Dayton, Ohio, August 5–7, 1998, pp. 80–87.
[8] S.M. Bhandarkar, Y.S. Warke, A.A. Khombadia, Integrated parsing of compressed video, in: Proceedings of International Conference on Visual Information Management Systems, Amsterdam, The Netherlands, June 2–4, 1999, pp. 269–276.
[9] V. Bhaskaran, K. Konstantinides, Image and Video Compression Standards: Algorithms and Architectures, Kluwer Academic Publishers, 1995, pp. 161–194.
[10] A. Bilas, J. Fritts, J.P. Singh, Real-time parallel MPEG-2 decoding in software, Technical Report 516-96, Department of Computer Science, Princeton University, March 1996.
[11] S.S. Cheung, A. Zakhor, Automatic News Watcher, Technical Report, Video and Image Processing Laboratory, University of California, Berkeley, 1999.
[12] H. Ching, H. Liu, G. Zick, Scene decomposition of MPEG compressed video, in: Proceedings of SPIE Conference on Digital Video Compression: Algorithms and Technologies, San Jose, CA, vol. 2419, February 1995, pp. 26–37.
[13] N. Gamaz, X. Huang, Scene change detection in MPEG domain, in: Proceedings of IEEE Southwest Symposium on Image Analysis and Interpretation, 1998, pp. 12–17.
[14] D.L. Gall, MPEG: A video compression standard for multimedia applications, Communications of the ACM 34 (4) (1991) 46–58.
[15] A. Hampapur, R. Jain, T. Weymouth, Production model-based digital video segmentation, Journal of Multimedia Tools and Applications 1 (March) (1995) 1–38.
[16] Y. He, I. Ahmad, M.L. Liou, A software-based MPEG-4 encoder using parallel processing, IEEE Transactions on Circuits and Systems for Video Technologies 8 (7) (1998) 909–920.
[17] P.R. Hsu, H. Harashima, Detecting scene changes and activities in video databases, Proceedings of IEEE International Conference on Acoustics, Speech, Signal Process 5 (April) (1994) 33–36.
[18] P.R. Hsu, H. Harashima, Spatiotemporal representation of dynamic objects, in: Proceedings of IEEE International Conference on Computer Vision and Patterm Recognition, 1993, pp. 14–19.
[19] A.K. Jain, A. Vailaya, X. Wei, Query by video clip, ACM Journal of Multimedia Systems 7 (5) (1999) 369–384.
[20] H. Jiang, A. Helal, A.K. Elmagarmid, A. Joshi, Scene change detection techniques for video database systems, ACM Journal of Multimedia Systems 6 (3) (1998) 186–195.
[21] J.R. Kender, B.L. Yeo, Video scene segmentation via continuous video coherence, in: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, June 1998, pp. 367–373.
[22] V. Kobla, D. Doermann, K.I. Lin, C. Faloutsos, Compressed domain video indexing techniques using DCT and Motion Vector information in MPEG video, Proceedings of SPIE Conference on Storage and Retrieval for Image and Video Databases V 3022 (1996) 200–211.
[23] V. Kobla, D. Doermann, K.I. Lin, C. Faloutsos, Compressed domain video indexing techniques using DCT and motion vector information in MPEG video, Proceedings of SPIE Conference on Storage and Retrieval for Image and Video Database V 3022 (February) (1997) 200–211.
[24] I. Koprinska, S. Carrato, Video segmentation of MPEG compressed data, in: Proceedings of ICECS, Lisboa, Portugal, September 7–10, 1998.

[25] I. Koprinska, S. Carrato, Detecting and classifying video shot boundaries in MPEG compressed sequences, in: Proceedings of EUSIPCO-98, Rhodes, Greece, September 1998.
[26] R. Lienhart, W. Effelsberg, R. Jain, VisualGREP: A systematic method to compare and retrieve video sequences, Proceedings of SPIE Conference on Storage and Retrieval for Image and Video Databases VI 3312 (1997) 271–282.
[27] K. Mayer-Patel, A parallel software-only video effects processing system, Ph.D. Dissertation, Computer Science, University of California, Berkeley, December 1999.
[28] H.J. Meng, D. Zheng, S.F. Cheng, Searching and editing MPEG compressed video in a distributed online environment, ACM Journal of Multimedia Systems 7 (4) (1999) 282–293, July.
[29] J. Meng, Y. Juan, S.F. Chang, Scene change detection in a MPEG compressed video sequence, Proceedings of SPIE Conference on Digital Video Compression: Algorithms and Technologies 2419 (February) (1995) 14–25.
[30] [MPEG 1] ISO/IEC 11172-2, Information Technology—Coding of Moving Pictures and Associated Audio for Digital Storage Media at upto about 1.5 Mbit/s—Video, Geneva, 1993.
[31] [MPEG 2] ISO/IEC JTC1/SC29/WG11/702 Revised, Information Technology—Generic Coding of Moving Pictures and Associated Audio, Recommendation H.262, Draft Intl. Standard, March 1994.
[32] A. Nagasaka, Y. Tanaka, Automatic video indexing and full-video search for object appearances, in: Proceedings of IFIP TC2/WG2.6 2nd Working Conference on Visual Database Systems, September 30–October 3, 1991, pp. 113–127.
[33] C.W. Ngo, T.C. Pong, R.T. Chin, Detection of gradual transitions through temporal slice analysis, in: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, Fort Collins, CO, June 23–25, 1999, pp. 36–41.
[34] K. Otsuji, Y. Tonomura, Projection detecting filter for video cut detection, in: ACM Intlernational Conference on Multimedia, August 1993, pp. 251–257.
[35] N. Patel, I.K. Sethi, Compressed video processing for cut detection, IEE Proceedings—Vision, Image and Signal Processing 143 (3) (1996) 315–323.
[36] N. Patel, I.K. Sethi, Video shot detection and characterization for video databases, Pattern Recognition 30 (4) (1997) 583–592.
[37] L.A. Rowe, K. Patel, B.C Smith, Performance of a software MPEG video decoder, Proceedings of ACM International Conference on Multimedia 34 (4) (1991) 46–58.
[38] Y. Rui, T.S. Huang, S. Mehrotra, Constructing table-of-content for videos, ACM Journal of Multimedia Systems 7 (5) (1999) 359–368.
[39] H. Sawhney, J. Hafner, Efficient color histogram indexing, in: Proceedings of IEEE International Conference on Image Processing, Austin, TX, November 13–16, 1994, pp. 66–70.
[40] I.K. Sethi, N. Patel, A statistical approach to scene change detection, in: Storage and Retrieval for Image and Video Databases, SPIE, vol. 2420, February 1995, pp. 329–338.
[41] B. Shararay, Scene change detection and content-based sampling of video sequences, in: Digital Video Compression: Algorithms and Technologies, SPIE, vol. 2419, February 1995, pp. 2–13.
[42] K. Shen, L.A. Rowe, E.J. Delp, A parallel implementation of an MPEG-1 encoder: faster than real time, in: Proceedings of SPIE Conference on Digital Video Compression: Algorithms and Technologies, San Jose, CA, February 1995.
[43] K. Shen, E.J. Delp, A fast algorithm for video parsing using MPEG compressed sequences, in: Proceedings of IEEE International Conference on Image Processing, October 23–26, 1995, Washington DC, pp. 252–255.
[44] B. Shen, I.K. Sethi, Convolution-based edge detection for image/video in block DCT domain, Journal of Visual Communication and Image Representation 7 (4) (1996) 411–423.
[45] B. Shen, I.K. Sethi, Block-based manipulations of transformed-compressed images and videos, ACM Journal of Multimedia Systems 6 (2) (1998) 113–124, April.
[46] S.M. Song, T.H. Kwor, W.M. Kim, H. Kim, B.D. Rhee, Detection of gradual scene changes for parsing of video data, Proceedings of SPIE Conference Storage and Retrieval for Image and Video Databases IV 3312 (1997).
[47] D. Swanberg, C.F. Shu, R. Jain, Knowledge guided parsing in video databases, in Storage and Retrieval for Image and Video Databases, SPIE, vol. 1908, 1993, pp. 13–24.

[48] G.K. Wallace, The JPEG still picture compression standard, Communications of the ACM 34 (4) (1991) 30–44.
[49] B.L. Yeo, B. Liu, Rapid scene analysis on compressed video, IEEE Transactions on Circuit and Systems for Video Technology 5 (6) (1995) 533–544.
[50] B.L. Yeo, On fast microscopic browsing of MPEG-compressed video, ACM Journal of Multimedia Systems 7 (4) (1999) 269–281.
[51] R. Zabih, J. Miller, K. Mai, A feature-based algorithm for detecting and classifying production effects, ACM Journal of Multimedia Systems 7 (2) (1999) 119–128, March.
[52] H.J. Zhang, A. Kankanhalli, S.W. Smoliar, Automatic partitioning of full-motion video, ACM Journal of Multimedia Systems 1 (1) (1993) 10–28.
[53] H.J. Zhang, S.W. Smoliar, Content-based video indexing and retrieval, IEEE Multimedia 1 (2) (1994) 62–72.
[54] H.J. Zhang, C.Y. Low, S.W. Smoliar, Video parsing and browsing using compressed data, Journal of Multimedia Tools Applications 1 (1) (1995) 89–111.