PARALLEL PARSING OF MPEG VIDEO

Suchendra M. Bhandarkar and Shankar R. Chandrasekaran

Department of Computer Science, The University of Georgia
Athens, Georgia 30602–7404, U.S.A.
Email: suchi@cs.uga.edu, shanchan@cisco.com

ABSTRACT

Video parsing refers to the detection and classification of abrupt and gradual scene changes in a video stream and constitutes an important preprocessing step in applications that treat video streams as sources of information. Parallel processing is proposed as a means of dealing with the high computational demands of video parsing. Parallel versions of two algorithms that detect scene transitions in compressed video streams are proposed. Three granularities of parallelism are investigated; Group of Pictures (GOP), frame and slice. Results show that the GOP-level implementation, which represents the coarsest granularity of task and data decomposition, always performs the best. The slice and frame levels of granularity take the second and third place respectively. The speedup is shown to be almost linear in the case of the GOP level of granularity, whereas the synchronization overheads are seen to be high for the frame and slice levels of granularity.

1. INTRODUCTION

With the emergence of *multimedia information systems*, videos are being increasingly looked upon as vital sources of information. One of the greatest problems with emerging multimedia technologies is the difficulty of automatically and reliably extracting "key" information from images, video and audio streams which could then be used for rapid browsing, navigation, indexing and content-based retrieval of the relevant information [16]. A great deal of current research effort has been devoted to video parsing or automatic scene change detection to automatically 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 and is a crucial first step that precedes domain–dependent analysis of the video [17]. Some of the domain–independent features of interest in a video stream include scene cuts, scene dissolves, fades, pans and zooms. A scene cut is an abrupt scene change whereas pans, dissolves, fades and zooms represent gradual scene changes in the video stream.

Video streams are often compressed for efficient transmission and storage. Scene detection techniques that operate directly on compressed video data 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 [6]. The MPEG-1 video compression standard relies on two basic techniques for compression: block-based motion compensation for the reduction of temporal redundancy and Discrete Cosine Transform(DCT)-based compression for the reduction of spatial redundancy. *Intracoded* (or I) frames in the MPEG-1 video stream use only DCT-based compression i.e., compression without motion compensation. Motion vectors are defined for each 16 $\times$ 16 pixel region of the frame, called a *macroblock. Predictive coded* (or P) frames, have macroblocks that are *motion compensated* with respect to a reference frame (I or P frame) in the immediate past (i.e., causal prediction). *Bidirectionally coded* (or B) frames, have macroblocks that are motion compensated with respect to, either a reference frame in the immediate past, immediate future or both (i.e., noncausal prediction or interpolative coding). The difference signal (or prediction error) is compressed using spatial redundancy reduction techniques based on the DCT and is transmitted along with the rest of the spatial information. The motion information is compressed using a variable-length entropy code to achieve maximum compression efficiency.

When decoding the MPEG-1 stream, I frames can be decoded independently. Decoding of the P frames requires the previously decoded reference frame (I or P) to be available whereas decoding of the B frames requires either one or two reference frames (from the past or future) to be available. So typical decoders buffer up to three frames i.e., the two 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 (GOPs). 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 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 of one or more macroblocks. A macroblock can be decomposed into pixel-level blocks. A macroblock is a 16 $\times$ 16 motion compensation unit whereas a block is a 8 $\times$ 8 DCT unit. This hierarchy is exploited in task de-

composition and task distribution during the parallelization process.

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. 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 consists of three sub-frames which are termed as the Y, U and V frames. The Y frame represents the brightness or luminance component of the original frame and the U and V frames represent the color or chrominance components. A DC image of a frame comprises of 3 DC sub-images corresponding to Y,U and V sub-frames.

The parallel 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 machine with multiple processors. Our implementation uses the shared memory model on account of its relative ease of programming abstraction and realization. For a small number of processors, parallel implementations on shared memory systems are typically faster than those on distributed memory systems.

2. RELATED WORK

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. MPEG-1 encoding with its inherent dependencies and other nuances poses several challenges to parallelization.

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, gray level/color histograms or gray level/color statistics computed for each frame in the video stream [15], or based on motion discontinuities computed using temporal filtering [10]. Feature-based approaches compute the difference in features such as edges between successive frames of the uncompressed video [14]. In some cases, a domain–specific video production model is used to guide the parsing process [7].

A number of video parsing techniques that entail minimal decompression of the video stream are based on the analysis of DC images [12, 17]. These techniques typically exploit the chrominance and/or luminance values of the DC images or their histograms to detect abrupt and gradual scene changes in the compressed video stream. The key frames derived from the scene change detection algorithm are subject to a clustering and matching process to generate a high-level representation such as a scene transition graph which can be used for rapid browsing and navigation of the video data [13]. Ngo *et al.* [9] use 1-D 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 2-D 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. Algorithms for computing the relative motion between successive frames to detect abrupt and gradual scene changes in a compressed video stream have been proposed in [3]. However, their major drawback is that it is not always possible to determine the motion vectors in the MPEG-1 video stream [3]. Consequently, Bhandarkar *et al.* [4] have presented parsing algorithms that 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 images performs better than the individual methods [4].

MPEG-1 encoding and decoding have received the most attention from the viewpoint of parallel processing [2, 5, 8, 11]. Common implementation platforms include shared memory multiprocessors [5], distributed memory multiprocessors [1, 8, 11] and a network of workstations [2]. The modes of parallelism that are commonly used include (a) functional parallelism where the video effects to be detected 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 each processor works on the entire set of frames assigned to it, and (c) spatial parallelism where regions of video frames are assigned to individual processors. As of date, we are not aware of any published work on parallel MPEG-1 video parsing. As video parsing is closely related to video decoding, it too should benefit from parallelism.

3. PARALLEL PARSING OF MPEG-1 VIDEO

Two different approaches to parallel MPEG-1 video parsing are implemented and analyzed in this paper. The first approach, termed as *Approach 1*, is based on [4] and uses DC images and motion vectors to detect scene changes. The DC images corresponding to the Y, U and V images of each frame in the MPEG-1 stream are generated using the method suggested in [12] without entailing complete decompression. Abrupt scene changes (i.e., scene cuts) are detected by thresholding the difference between the sums of the DC values from the Y, U and V images of two consecutive frames. In the event of an abrupt 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 [12].

When parsing the video 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 determined by first computing the *motion distances* between every pair of successive frames [3]. 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 [3]. During a camera pan, a majority of the motion vectors are aligned in the direction of the pan [4]. During a zoom-in or zoom-out, a majority of the motion vectors either point outward or inward, respectively [4]. Dissolves are detected based on the observation that the prediction error values in the encoded macroblocks are high [3].

The second approach, termed as *Approach 2*, is based on [9] 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 the stitched images. The shapes of the boundaries between segments reveal the nature of the scene transitions. Abrupt scene cuts and gradual scene changes are detected by thresholding the difference sequences. Detection of pans, zoom-ins and zoom-outs is much more complex and hence not performed.

The two approaches discussed above are parallelized in a parallel video parsing system which is similar to the parallel MPEG decoding system discussed in [5]. The system consists of a main process, a group of worker processes and a display process. The main process scans the input MPEG-1 stream, prepares the units of parallel computation (i.e., tasks) and enqueues them in a circular queue in shared memory. The workers dequeue the tasks, process them and convey the intermediate and final results to the display process. The relevant information consisting of the DC images and the motion vectors is extracted locally within the tasks assigned to the workers and is shared with the other workers in order to compute the final results. All the processes use shared memory for communication and synchronization. The main process does not perform any computation, but monitors the MPEG-1 stream for the beginning and end of a frame, slice or a particular sequence of frames before enqueuing tasks in the 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. GOPs, frames and slices are chosen as units for task decomposition in the current implementation. The sequence layer is not chosen because many available MPEG-1 videos typically have very few sequences; most have just one. The macroblock layer is not practical because of the high number of inter-macroblock dependences within a slice which, in turn, would entail a high inter-process synchronization overhead. In the current implementation, only three frames and their related information are resident in memory at any point in time. The three frames resident in memory include the two reference frames and the frame being currently decoded or analyzed. The related information comprises of the DC images, motion vectors, and other assorted data needed for the working of the system. The parallel algorithms are implemented on a quad-processor shared-memory machine running Solaris. Interprocess communication is done solely through shared memory.

3.1. Parallel video parsing at the GOP level

At the GOP level, each worker process tackles a distinct GOP and processes the results. A GOP is deemed closed if it can be decoded without reference to frames in any other GOP. Since the MPEG-1 encoder typically transmits frames out of their temporal sequence, the resulting GOPs are not closed. GOPs that are not closed entail communication and synchronization overhead among the worker processes. Thus, the master process has to preprocess the MPEG-1 stream to ensure that the resulting GOPs are closed. The resulting closed GOPs are termed as custom GOPs. With custom GOPs, the communication and synchronization overhead among the worker processes is minimal. Most of the synchronization and communication is between the main process and the worker processes. In the remainder of the paper, the term GOP implies a custom GOP unless stated otherwise.

Approach 1: The main process inserts the custom GOPs along with relevant header information into the task queue. The workers dequeue the tasks from the queue and process them independently i.e., with no communication between the workers. Each worker process uses the algorithms in [3, 4, 12] for detecting scene cuts, pans, zooms and gradual scene changes using the DC images and motion vector information 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. Access to the task queue is synchronized. The workers access the task queue until it is empty.

The final results are computed in two steps. In the first step, the worker extracts the three DC images for each frame in its GOP, and computes the sum of the DC values for each of the three images. In the second step, the difference between the sums of successive frames or frames that are k apart (for dissolve detection) is computed. 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. The workers compute the motion vectors within their assigned GOPs followed by the computation of the motion distance with respect to a reference frame within the GOP.

Approach 2: Each worker locally stitches the 1-D strips from the three DC images of each frame in its GOP to form a 2-D image. The scene changes are detected by segmenting the 2-D image using an edge detector in the YUV feature space with synchronization between workers at the GOP

boundaries. Edge detection is done while extracting the 1-D strips, thus saving memory since storing the entire 2-D image in memory is not necessary. The DC sums of each strip are stored in a shared address space. Scene cuts and pans use the differences between consecutive frames whereas dissolves use the k-difference method of Approach 1.

3.2. Parallel video parsing at the frame level

At the frame level, each worker tackles a single frame at a time resulting in a finer-grained task and data decomposition compared to the GOP level. Computing the differences in the DC and motion vector values between successive frames results in excessive synchronization and communication overhead between the worker processes. To minimize this 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 and communication. However, the inherent dependences 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.

Approach 1: The main 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 workers dequeue the tasks and process the frames in the temporal order of the frame numbers on account of the inter-frame dependences. 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 processed. The above two conditions enforce a global ordering on the computation. A worker, not processing a frame, sleeps on a condition variable.

The workers compute the results in two steps. In the first step, a worker process computes the sum of the DC values for its assigned frame and records it in the shared memory. The first step is local to each worker and does not entail any synchronization. The second step involves synchronization among the workers after every I frame in the MPEG-1 stream. After every I frame, the workers use the intermediate information to compute the final results. The worker that processes an I frame, sets a global variable that signals all the other workers to suspend their tasks. After having processed the I frame, the worker awakens the other sleeping workers. Since the global variable is set, all the workers halt the processing of future frames and proceed to compute the final results. They divide the preprocessed work among themselves and compute the inter-frame differences. The computation of the motion vectors is performed in a similar manner. The reference frames that are resident in shared memory are accessed synchronously by the workers to compute the motion vectors and the motion distances.

Approach 2: The main process for Approach 2 is the same as that for Approach 1. The workers extract the three strips, horizontal, vertical and diagonal, from the three DC images of the frames that they are currently processing. The processing of the frames follows the same order as in Approach 1. At the end of each frame, the workers synchronize to compute the pixel differences. Each worker 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.

3.3. Parallel video parsing at the slice level

At the level of individual slices, each worker is assigned a slice to be processed. This represents the finest granularity of task and data decomposition in our implementation. 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 processing 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 and the final results computed during synchronization after every I frame in the MPEG-1 stream. As before, the inherent dependences 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. At least two circular task queues are used in this implementation. The main 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 enqueue operation performed by the main process and the dequeue operations performed by the worker processes to be overlapped without entailing a high synchronization overhead.

Approach 1: The main 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 task queue. Processing takes place in the temporal order of the frame numbers. However, the slices within a frame are processed in any order. The workers in possession of 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 worker process in the slice-level implementation

is very similar to its frame-level counterpart except for the fact that at the slice-level, a frame is processed by multiple workers whereas at the frame-level, a frame is processed by a single worker. The worker processes compute the necessary intermediate results and synchronize at the end of every frame. The main and worker processes are very similar in both cases i.e., whether the DC values or the motion vectors are used to detect scene changes.

Approach 2: The main process in Approach 2 is identical to the one in Approach 1. 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 and computes the inter-frame differences needed to detect scene cuts and dissolves. A more parallel version should utilize all workers in the computation of the sums and differences, but would be highly complex.

4. PERFORMANCE EVALUATION

The performance of the parallel video parsing schemes discussed in the previous section were analyzed in terms of speedup, synchronization overhead and general memory requirements. All our experiments were conducted on a Sun E450 which is a quad-processor, shared-memory multiprocessor machine running Solaris in conjunction with the Solaris 2.5 shared memory library. All the processes in the system were traditional UNIX processes with shared memory used as the medium of inter-process communication.

4.1. Parallel parsing at the GOP level

Approach 1: Speedup curves for the parallel MPEG-1 parsing at the GOP-level with Approach 1 using DC images and task queue size = 9 are shown in Figure 1(a). The speedup is computed with respect to the time taken by a single worker system. Since the Sun E450 has only 4 processors, the speedup increases linearly with an increasing number of worker processes up to 3 and then levels off. If more than 3 worker processes are executed, the processes are time-shared on the processors since the master process is assigned to one of the processors. Several different sizes for the task queue (with 3 worker processes) were examined. Since the time taken to put a GOP into the queue is much less than the time taken to process a GOP, for task queue sizes ≥ 3, the workers do not spend any time waiting for the task queue to be filled. Hence the speedup was found not to vary for task queue sizes ≥ 3. Since the GOP level implementation entails minimal synchronization, the speedup was seen to be almost linear. However, the GOP size (i.e., the number of frames in the GOP) was seen to affect the load imbalance among the worker processes and hence the speedup. Larger GOP sizes caused greater load imbalance and greater deterioration in the speedup. The disparity in the speedup values, was seen to increase with an increase in the number of worker processes. This could be attributed to fact that the initial wait time (i.e., time taken to fill the task queue for the first time) increases with an increase in the number of worker processes.

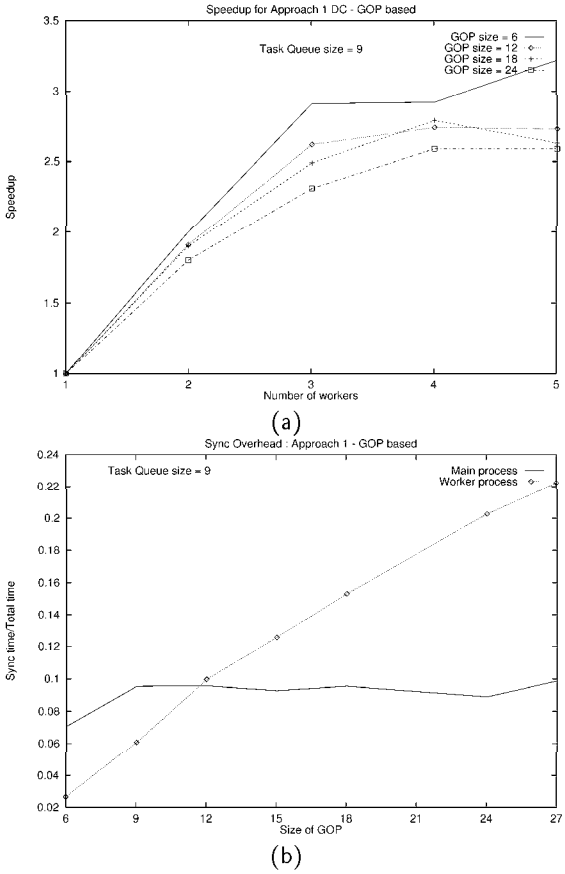

Figure 1: (a) Speedup curves and (b) synchronization overhead for the GOP-level implementation with Approach 1 using DC images and task queue size = 9

The synchronization overhead was measured as the percentage of the total execution time that is spent in synchronization primitives due to stalls resulting from access to shared data (including the task queue) and stalls resulting from the data dependences. Figure 1(b) shows the synchronization overhead for both, the main process and a worker process for a system with 1 main process and 3 worker processes. The initial wait time increases with the GOP size causing the synchronization overhead to increase with GOP size.

Approach 2: Speedup curves for the GOP-level implementation with Approach 2 using DC image strips are shown in Figure 2(a) for different GOP sizes. As in the case of Approach 1, the smaller-size GOPs perform better than the larger-size GOPs. As Figure 2(b) shows, the initial wait time experienced by the worker processes increases with the GOP size resulting in an increase in the synchronization overhead. For the main process, however, the synchronization overhead decreases with increase in GOP size, becoming almost negligible for GOP sizes greater than 12 frames.

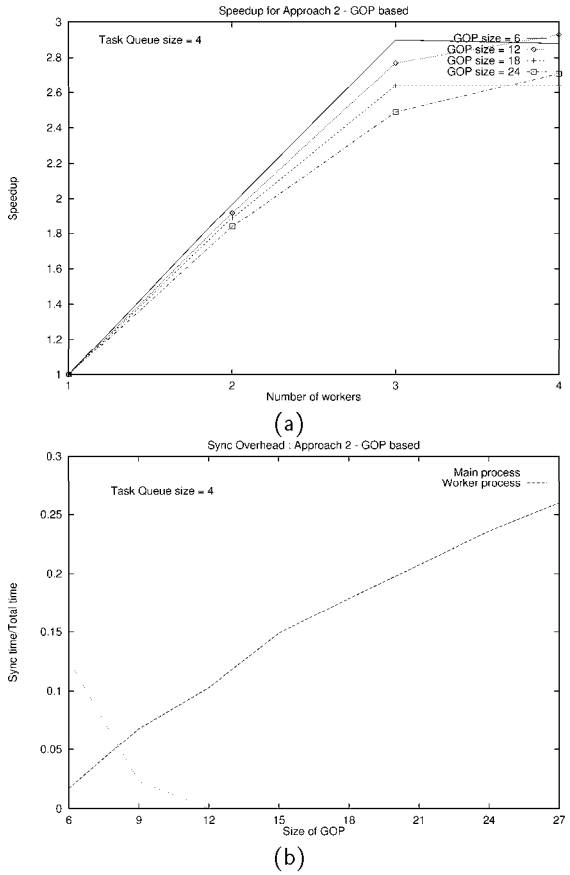

Figure 2: (a) Speedup curves, and (b) synchronization overhead curves for the GOP-level implementation with Approach 2 using DC image slices and task queue size = 4

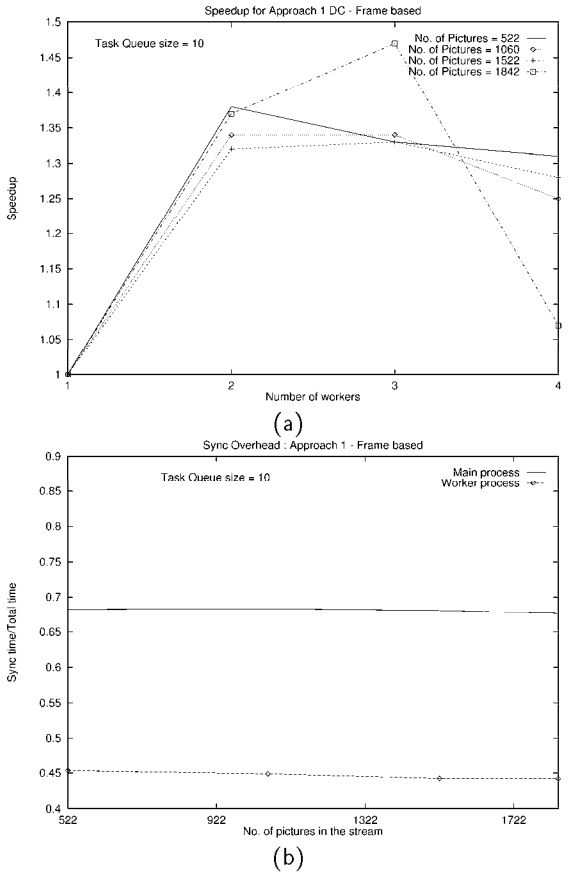

Figure 3: (a) Speedup curves and (b) synchronization overhead for the frame level implementation with Approach 1 using DC images and task queue size = 10

4.2. Parallel parsing at the frame level

Approach 1: Figure 3(a) shows the speedup curves for the frame level implementation with Approach 1 which uses DC images. Several different sizes for the task queue were examined. The speedup was observed to be sub-linear because of high synchronization overhead which limits parallelism. The speedup degrades drastically as the number of worker processes exceeds 3. The size of the task queue made no noticeable difference. Load imbalance was not a significant factor because of the finer granularity of task and data decomposition.

The frame-level implementation involves high synchronization overhead. The overhead, measured as the percentage of total time spent in synchronization, was seen to be constant for both, the main process and the worker processes. The initial wait time is present but is overshadowed by the enormous amount of subsequent synchronization. The synchronization overhead is independent of the number of frames in the MPEG-1 stream because every frame is associated with the same amount of processing and synchronization overhead. As Figure 3(b) shows, the synchronization overhead is about 67.5% for the main process and 45% for a worker.

Approach 2: Figure 4(a) depicts the speedup for the frame-level implementation using DC image slices. The speedup was seen to be sub-linear because of the high synchronization which limits the extent of parallelism. For more than 3 workers, the speedup dropped to less than 1, implying thereby that the communication overhead exceeded the gain from parallelism. This approach involved high synchronization overhead as can be seen in Figure 4(b). The synchronization overhead, was constant for both the main process (66%) and the worker processes (46%) and independent of the number of frames in the MPEG-1 stream since each frame is subject to the same amount of processing and synchronization overhead.

4.3. Parallel parsing at the slice level

Approach 1: Experiments were conducted for several different frame resolutions (i.e., number of slices in a frame) and task queue sizes. The size of a single slice, however, is a constant. As Figure 5(a) shows, the slice-level implementation yielded better speedup than the frame-level implementation but worse than the GOP-level implementation. The slice-level implementation is constrained by the same synchronization overhead as its frame-level counterpart. The improvement over the frame-level implementation could be

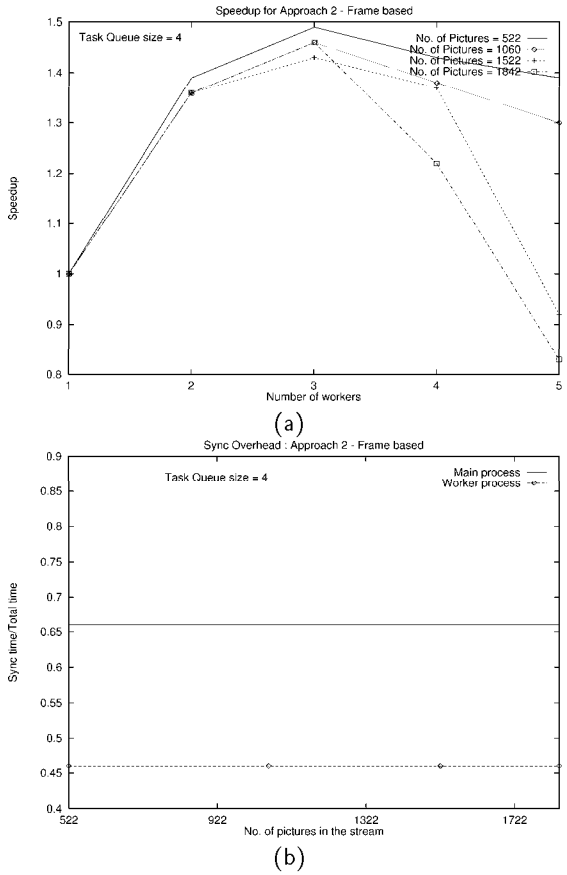

Figure 4: (a) Speedup curves and (b) synchronization overhead for the frame-level implementation with Approach 2 using DC image slices and task queue size = 4

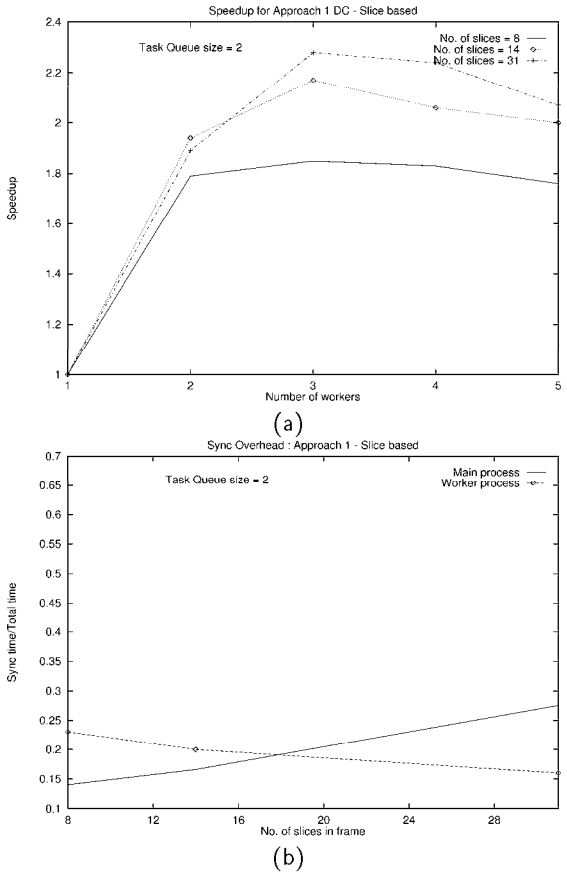

Figure 5: (a) Speedup curves and (b) synchronization overhead for the slice-level implementation with Approach 1 using DC images and task queue size = 2

attributed to the parallel processing of a frame by several worker processes. In Figure 5(a), test results are shown for three different frame resolutions. The performance was seen to be better for larger frame resolutions due to the relatively infrequent synchronization by the workers. Load imbalance was not a major issue because of the fine granularity of task and data decomposition. Figure 5(b) depicts the synchronization overhead. For a worker process, the synchronization overhead was seen to decrease with an increase in the frame resolution. Since there are more slices in a frame, workers spend more time in computation between synchronization and hence synchronize less frequently. The main process was seen to spend more time in synchronization with an increase in frame resolution.

Approach 2: As Figure 6(a) shows, the speedup was observed to be better for higher frame resolutions. For both, the main process and the worker processes, the synchronization overhead was seen to decrease with increasing frame resolution (Figure 6(b)).

4.4. Memory requirements

The system wide memory requirements are depicted in Figure 7 for the GOP-level, frame-level and slice-level implementations. The GOP-level implementation was the most memory intensive with the memory requirement increasing with the GOP size, task queue size and number of worker processes. For the frame-level implementation, the memory requirement was seen to be less stringent with the task queue being the main contender. However, since each slot in the task queue holds a single frame it was less memory intensive than its GOP-level counterpart. The slice-level implementation was the least memory intensive since each each slot in the task queue holds only a single slice. In the frame-level and slice-level implementations, the memory requirement was seen to be less dependent of the number of workers since the three frames resident in shared memory are common to all the workers.

5. CONCLUSIONS AND FUTURE WORK

A parallel parsing system for MPEG-1 video streams was discussed, implemented and evaluated. Three levels of granularity for task and data decomposition were investigated, the GOP level, frame level and slice level. The GOP-level implementation which represents the coarsest granularity of parallelism yielded the best speedup and scalability characteristics followed by the slice-level and frame-level im-

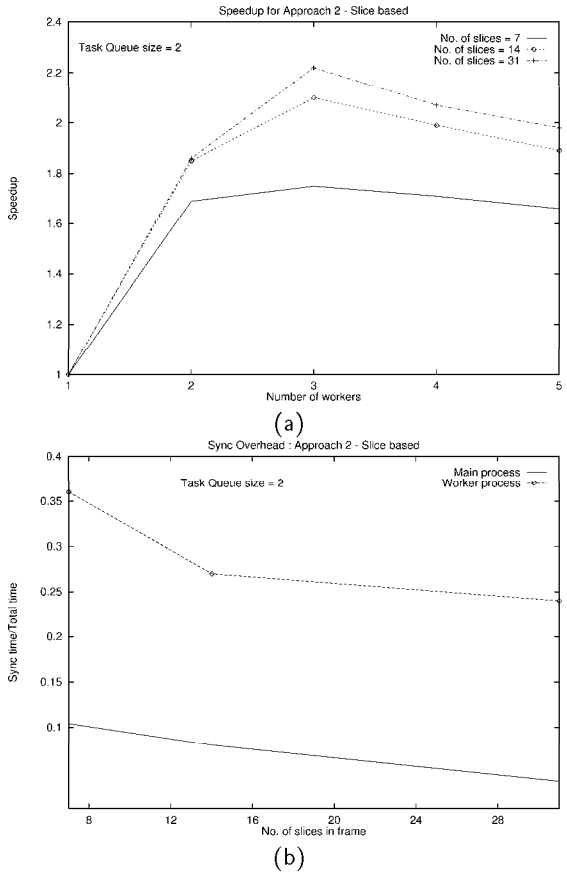

Figure 6: (a) Speedup curves and (b) synchronization overhead for for the slice-level implementation with Approach 2 using DC image slices and task queue size = 2

plementations. 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 machine. More advanced video coding standards such as MPEG-4 and MPEG-7 will also be investigated.

6. REFERENCES

[1] I. Agi and R. Jagannathan, A portable fault-tolerant parallel software MPEG-1 encoder, *Proc. Second IASTED/ISMM Intl. Conf. Distributed Multimedia Systems and Applications*, Stanford, CA, Aug. 1995.

[2] S. M. Akramullah, I. Ahmad and M. L. Liou, Parallel MPEG-2 encoder on ATM and Ethernet connected workstations, *Lecture Notes in Computer Science*, Vol 1557, Springer-Verlag, Berlin, pp. 572-574, 1999.

[3] S. M. Bhandarkar and A. A. Khombadia, Motion based parsing of compressed video, *Proc. IEEE Intl. Workshop on Multimedia Database Management Systems*, Dayton, Ohio, August 5-7, 1998, pp. 80-87.

[4] S. M. Bhandarkar, Y. S. Warke and A. A. Khombadia, Integrated parsing of compressed video, *Proc. of Intl. Conference on Visual Information Management Systems*, Amsterdam, The Netherlands, June 2-4, 1999, pp. 269-276.

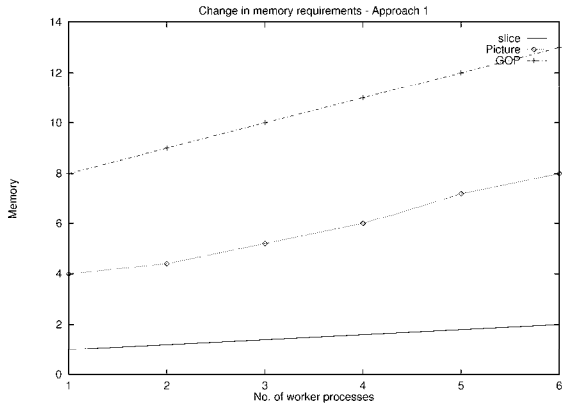

Figure 7: Memory requirements for the three levels of parallel processing granularity: GOP, frame and slice

[5] A. Bilas, J. Fritts and J. P. Singh, Real-time parallel MPEG-2 decoding in software, Technical Report 516-96, Department of Computer Science, Princeton University, March 1996.

[6] D.L. Gall, MPEG: A video compression standard for multimedia applications, *Communications of the ACM*, Vol. 34, No. 4, April 1991, pp. 46–58.

[7] A. Hampapur, R. Jain and T.Weymouth, Production model–based digital video segmentation, *Jour. Multimedia Tools and Applications*, Vol. 1, March 1995, pp. 1–38.

[8] Y. He, I. Ahmad and M. L. Liou, A software based MPEG-4 encoder using parallel processing, *IEEE Trans. on Circuits and Systems for Video Technologies*, Vol 8, No 7, pp. 909-920, 1998.

[9] C. W. Ngo, T. C. Pong and R. T. Chin, Detection of gradual transitions through temporal slice analysis, *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, Fort Collins, Colorado, June 23-25, 1999, pp. 36-41.

[10] B. Shararay, Scene change detection and content–based sampling of video sequences, *Digital Video Compression: Algorithms and Technologies*, SPIE Vol. 2419, Feb. 1995, pp. 2–13.

[11] K. Shen, L. A. Rowe and E. J. Delp, A parallel implementation of an MPEG-1 encoder: faster than real time, *Proc. of SPIE Conf. Digital Video Compression: Algorithms and Technologies*, San Jose, CA, Feb 1995.

[12] B. L. Yeo and B. Liu, Rapid scene analysis on compressed video, *IEEE Trans. Circuit and Systems for Video Technology*, Vol. 5, No. 6, 1995, pp. 533-544.

[13] B. L. Yeo, On fast microscopic browsing of MPEG-compressed video, *ACM Jour. Multimedia Systems*, Vol. 7, No. 4, July 1999, pp. 269-281.

[14] R. Zabih, J. Miller and K. Mai, A feature based algorithm for detecting and classifying production effects, *ACM Jour. Multimedia Systems*, Vol. 7 No. 2, March 1999, pp. 119-128.

[15] H.J. Zhang, A. Kankanhalli, and S.W. Smoliar, Automatic partitioning of full-motion video, *ACM Jour. Multimedia Systems*, Vol. 1, No. 1, 1993, pp. 10 28.

[16] H.J. Zhang and S.W. Smoliar, Content–based video indexing and retrieval, *IEEE Multimedia*, Vol. 1, No. 2, Summer 1994, pp. 62–72.

[17] H. J. Zhang, C. Y. Low and S. W. Smoliar, Video parsing and browsing using compressed data, *Journal of Multimedia Tools Applications*, Vol. 1 No. 1, 1995, pp. 89-111.