

Parallel Algorithms for MPEG Video Compression with PVM

S. Bozóki*, S. J. P. Westen**, R. L. Lagendijk** and J. Biemond**

*Dept. of Telecommunications, Technical University of Budapest
H-1111 Budapest, Sztocek u. 2., Hungary
e-mail: bozoki@com.hit.bme.hu

**Information Theory Group, Dept. of Electrical Engineering
Delft University of Technology, P.O. Box 5031, NL-2600 GA, Delft, The Netherlands
e-mail: stefan@it.et.tudelft.nl

Abstract

This paper presents parallel algorithms for MPEG video compression by using the PVM library. Because of the huge amount of computation, a sequential software encoder is slow at the coding of reasonably sized images (e.g. 0.1-0.3 frames per second). In order to speed up the process, we developed two kinds of parallel algorithm, which run on a cluster of workstations connected to the network. The first method encodes slices in parallel, while the second one encodes equally sized Groups Of Pictures (GOP) in parallel. The communication between the machines is facilitated by the PVM package. The reference sequential algorithm can encode a SIF sized sequence at 0.2 frames per second on a SUN Sparc 2 using the GOP pattern of IBBP. The slice level parallelization can take this rate up to nearly 0.7 frames per second with the use of 6 Sparc 2s connected via Ethernet. The degradation of the performance is a consequence of the low bandwidth of the network and the relatively high communication activity. The GOP level parallelization provides an encoding rate of nearly 1.0 frame per second at the same conditions. In this case, the communication is reduced significantly, however, the load balancing may be slightly worse if the size of the GOP is large ($N \geq 10$). Our parallel algorithms are entirely portable and produce bitstreams fully conforming the MPEG-1 standard. The methods can be applied to MPEG-2 encoding as well. The system performs nearly as well as the parallel Berkeley encoder by using the second algorithm. However, it produces a video stream of better quality because the encoding is done correctly.

Keywords: MPEG; video compression; PVM; load balancing; parallel algorithm; message passing parallel system

1. Introduction

The MPEG video compression algorithm is the basis of many current and future communications and multimedia applications such as Video CD, CD-I with full motion (MPEG-1 [1]) or digital television, Video On Demand (VOD) and Digital Video Disc (DVD) (MPEG-2 [2]). The MPEG standards describe a syntax for the data and specify the decoding steps, however, they do not specify the encoder. Although MPEG encoders have been realized in hardware [3], a more flexible approach is to use software encoding. This solution has the advantage that in the design phase of the encoder, changes can be easily made and evaluated, and that in the production phase of the encoder, a modification requires a software update rather than a redesign of the hardware. Furthermore, a pure software encoder can perform a more sophisticated image sequence analysis in order to maximize the quality of the video material at a given bit-rate. This is especially important at low bit-rates, such as in the case of Video CD (1.15 Mbits/s). However, a sequential software encoder is slow at the coding of reasonably sized images (e.g. images in Source Input Format /SIF/, 352x288 or 352x240 pixels) even if a powerful workstation is used.

This is because the MPEG video compression requires a huge amount of computation. In particular the Discrete Cosine Transform (DCT), the Motion Estimation (ME) and Motion Compensation (MC) are computationally intensive. For instance, our speed-optimized sequential algorithm encodes a SIF sized sequence at 0.2 frames* per second on a SUN Sparc 2 if the whole set of algorithms is used.

The encoding rate can be increased by using a distributed parallel system. In our case this is a cluster of workstations connected to the network. The communication between the machines is facilitated by the Parallel Virtual Machine (PVM) library [4], which has become a widely used program package for managing tasks and communication between processors in a distributed computing environment. Unfortunately, even if we use powerful machines, the system might not achieve real time processing (encoding 25-30 frames per second) because several factors may degrade the performance**. Especially the bandwidth of the network and the I/O communications impose a critical burden. Since image sequences are generally read from computer files, either from the local disk or from the Network File System (NFS), the disk transfer time of a frame forms an upper boundary to the encoding rate. The fluctuation of the load on the network and on the machines also influences the performance of the system. In fact, real-time encoding is not always required, off-line processing is satisfactory for many applications (e.g. CD, VOD), but it should also be done within a reasonable amount of time. Taking everything into account, a good load balancing strategy is essential, while the communication overhead (number of communication start-ups, amount of data transferred) should be kept as low as possible.

Another way for implementing a parallel system is to use a supercomputer containing many processor nodes. In this case, even "faster than real-time" performance can be achieved as reported in [6]. However, an application written for a specific multiprocessor system is usually less portable.

We implemented two kinds of algorithm for MPEG encoding on the aforementioned parallel system. They provide different levels of parallelism, the first one processes a picture in parallel (spatial parallelization), while the second one processes equally sized groups of pictures in parallel (temporal parallelization). We will compare the performance of our algorithms with that of the public domain parallel Berkeley encoder [5]. It uses a similar level of parallelization as our second method, but the size of the groups may vary. Although it ensures an excellent load balancing, it should be noted that correct encoding is not always possible, an inequality may exist between the encoder and decoder. As a consequence, the quality of the decoded video is usually worse than that of produced by its reference sequential encoder. This is not the case with our parallel algorithms.

The developed algorithms are entirely portable and produce bitstreams fully conforming the MPEG-1 standard. The methods can be applied to MPEG-2 encoding as well. The software requires a PVM version of 3.3.x. The input image format is raw YUV. The current version has the limitation of producing a variable bit-rate (see Section 2) stream only. It is advantageous if the workstations are mounted to the NFS. Except for the Berkeley encoder, we do not know about any other public domain parallel MPEG implementation. The mentioned real-time software encoder is essentially a modified version of the Berkeley one.

The organization of this paper is as follows: Section 2 gives an overview of the MPEG video compression, Section 3 describes its time-consuming operations, and it discusses the parallelization possibilities, emphasising the dependencies in the coding algorithm. Section 4 presents the developed algorithms, while Section 5 contains the results. In Section 6 we discuss some optimization issues and our future work. The conclusion can be found in Section 7.

* An image is called frame if it contains all the lines. In contrast with this, a field contains only the even or the odd lines. Interleaving two fields of the opposite parity results in a frame.

** Throughout this paper performance degradation will mean if the speedup does not change, or even it gets worse by increasing the number of computing resources.

2. Overview of the MPEG video compression algorithm

An image is described by three components, a luminance (Y) and two chrominance (U and V) image planes. MPEG-1 and most of the modes in MPEG-2 use the so called 4:2:0 colour space, which means that the number of U and V samples is half of the number of Y samples in both the horizontal and vertical directions. A frame is divided into non-overlapping areas called macroblocks (MB). A macroblock contains a block of 16x16 pixels in the Y plane and a block of 8x8 pixels in both the U and V planes. The Y content is broken up to 4 blocks of 8x8 pixels (Figure 1).

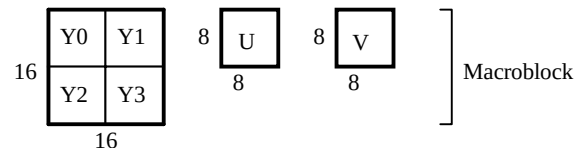


Figure 1. Components of the macroblock

MPEG video works well on natural image sequences which contain high spatial and temporal redundancies. Spatial redundancy means that the picture elements (pixels) are similar to each other within a certain area on a frame. This is utilized by a kind of frequency transform called *two-dimensional DCT*, which is applied to the 8x8 blocks of each macroblock, resulting in 64 coefficients per block. A small number of them is significantly larger than the others, the energy of a block is usually concentrated in the low frequency domain. After quantization a block can be represented by a few coefficients (e.g. 6...9), achieving a good compression in this way. This compression is improved further by assigning Variable Length Codes (VLC) to special symbols created from the coefficients. A macroblock coded in such a way is called *intra*, since it uses information only from itself. It can be reconstructed by the inverse procedure.

Temporal redundancy refers to the similarity among a certain number of consecutive frames. It makes possible that the images can be predicted from each other. An image, from which the prediction is made, is called *reference* frame. The prediction is enhanced by *motion compensation*, which is the use of motion information at the differential encoding. In MPEG video the motion information is found by *block-based motion estimation* by using the luminance component of the image. It works on all the macroblocks as follows: The Y content of a macroblock on the current frame is shifted over a search window on the reference frame, and a similar area of 16x16 pixels is searched for (Figure 2). The similarity is evaluated by means of a cost function at certain positions. Where the function has the minimum, that place is considered the best match. The relative translation of the macroblock is described by a motion vector. A set of motion vectors for an image is called *motion vector field*.

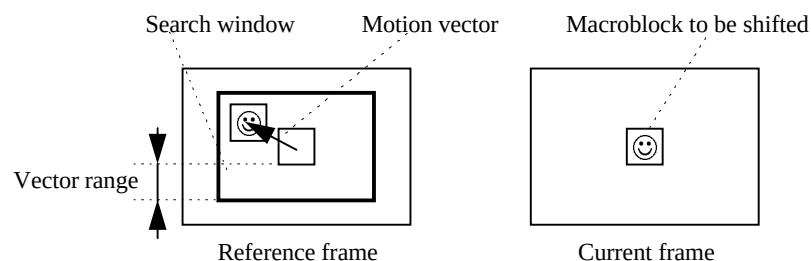


Figure 2. Block based motion estimation in MPEG video

If the best match is good enough, the corresponding pixels are subtracted from each other, that is to say, the actual macroblock is motion compensated. In MPEG they are called *predictive* as well. Since such a macroblock still shows some spatial redundancy, the aforementioned DCT coding is applied to the 8x8 blocks in it.

A decoder reconstructs a predictive macroblock by using the corresponding reference area accessed by the motion vector. However, in case of no good match the macroblock is coded as intra because it is likely to produce less bits.

Based on the above mentioned methods, MPEG video defines three picture types, namely I, P and B pictures. Figure 3 shows the relationships among them. I frames are *intra coded frames*, they are encoded without reference to other frames by utilizing only the spatial redundancy. Consequently, they consist of intra coded macroblocks only. P frames are *predicted frames*, they are predicted from the previous I or P frame by using motion compensation. B or *bi-directionally coded frames* are predicted from the previous and the next I or P frame. In this latter case two motion vectors are searched for each macroblock, one on the previous reference (past prediction) and the other on the next one (future prediction). The quality of the matching determines whether the prediction is made from the past or from the future or from both of them. In this latter case the corresponding reference areas are averaged. If there is no good match at all, the macroblock is coded as intra. As a consequence, P and B pictures may contain intra and predictive coded macroblocks. The basic idea behind the use of B pictures is that some areas may not be found correctly on either of references because objects can cover each other, or scene change can happen as well. Since a B frame uses future prediction, its future reference has to be transmitted before that frame. B pictures are never used as reference.

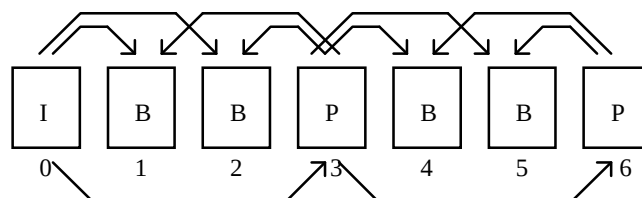


Figure 3. Picture types with dependencies in MPEG video

The above mentioned coding algorithm is used in MPEG-1. MPEG-2 is far more complicated as far as the prediction is concerned, but the basic ideas are the same. We should also emphasize that Figure 3 is only a typical image pattern in MPEG. In fact, there can be any number of P and B pictures between the reference ones. Moreover, their use is optional, a sequence may consist of I pictures only. I pictures provide access points where decoding can start. A fast random access can be facilitated by the use of many I pictures. However, their compression ratio is the least, typically 6...9. P pictures can be compressed better, usually by a factor of 12...20. B pictures provide the highest compression, typically 25...50. The overall compression ratio is between 20 and 30 in case of applications used in practice. All the values refer to high quality coding and are dependent on the image content. The more complex an image is, the more bits are needed to represent it.

Because of the characteristics of the encoding algorithm (different picture types, VLC) and the usually varying image content, the output bit-rate is fluctuating. However, most communication channels require a constant bit-rate (CBR) application, thus, the output of the encoder has to be buffered. Similar buffer is located at the input of the decoder. In order to prevent the buffers from over/underflow, the number of bits produced has to be controlled (rate control). This is usually done by changing the quantization factor on macroblock or on slice level. However, a variable bit-rate (VBR) stream having the same mean bit-rate usually has a better quality. In this case, a constant quantization stepsize is used throughout the sequence for each picture type.

An MPEG video stream has a layered structure. On the top there is the *Sequence Layer*, whose header contains general parameters for the sequence (e.g. frame dimensions, frame rate, bit-rate, etc.). The sequence consists of some *Groups of Pictures* (GOP) indicating the second layer. A GOP always starts with an I or B frame and ends with an I or P frame. For instance, Figure 3 can be considered a GOP. They offer random access in the sequence and facilitate the editability.

The next layer is the *Picture Layer*. A picture is divided into slices (*Slice Layer*) consisting of consecutive macroblocks. A slice is exactly a horizontal row of macroblocks in MPEG-2, while MPEG-1 allows greater flexibility for their position and size. Figure 4 shows a possible division. They are used for re-synchronization in the decoder if the bitstream is errored during transmission. The last two layers are the *Macroblock Layer* and the *Block Layer*.

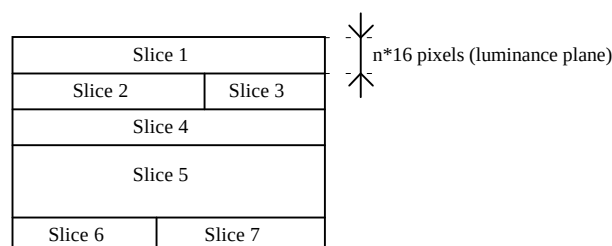


Figure 4. A possible division of a frame into slices (only in MPEG-1)

The MPEG-1 encoding is essentially used for optical recording of digital video at a bit-rate of 1.15 Mbits/s. In order to achieve a good quality, MPEG-1 starts with images in SIF. A SIF image satisfies the 4:2:0 format, having a size of 352x288 (or 352x240) pixels for the Y component and a size of 176x144 (or 176x120) pixels for the U and V components.

3. Parallelization of the encoding process

3.1 Computationally intensive operations

As it was mentioned in the introduction, some parts of the encoding process require a huge amount of computation. Therefore, it was essential to write an efficient, speed optimized sequential code, on which the parallel implementation could be based. We developed it ourselves without using any public domain software except for the DCT and its inverse (IDCT) routines. Its execution performance is not far from that of the Berkeley encoder which is the fastest portable public domain implementation.

The DCT is one of the most computationally intensive operations. Even using a very fast implementation like [7], the number of multiplications is over 450,000 and the number of additions is over 1,200,000 if it is applied to a SIF image. The motion estimation requires even more calculations, the computational complexity is determined by the size of the search window, i.e. the motion vector range (see Figure 2). Evaluating all the possible positions (full search) is a waste of time, although it always finds the best match, the number of bits produced for that macroblock is likely to be the least. However, the number of operations is over 200 millions on a SIF frame if the vector range is 16. We use a sort of three-level hierarchical search algorithms, which aims at finding the motion vectors on image planes of different resolutions. The search starts on a low resolution image, the results are initial displacements on a higher resolution level. The vectors are refined in each step at the reduction of the size of the search window. The number of operations is reduced by a factor of 50 about in this way, and the method provides a reliable vector field.

The motion compensation does not have so much contribution to the encoding time, but it should be considered as significant because the MB decision (how to code a given macroblock) is usually carried out by calculating MB variances.

Using the above mentioned algorithms, our sequential encoder is capable of coding SIF sized images at a rate of 0.2 frames per second on a SUN Sparc 2 workstation. The GOP pattern is IBBP, the motion vector range is 32 in each direction in this case. The encoding times are 2, 4, 6 seconds for I, P, B pictures respectively, excluding the I/O overhead. Figure 5 illustrates the contribution of the main coding steps to the encoding time of a P picture.

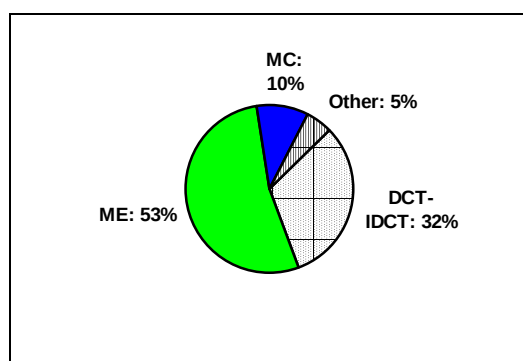


Figure 5. Time contribution of the coding tasks in a P frame
(Note: DCT-IDCT involves quantization and dequantization)

3.2 Dependencies in the coding algorithm

Because of the large number of calculations, it is advisable to do the encoding in parallel. The layered syntax offers several possibilities, but we should allow for some difficulties. The finest parallelism can be achieved on block level, since all the blocks are transformed and quantized independently. The macroblock level suggests a possibility to find the motion vectors for a frame independently. However, both levels of parallelism would be too fine for such a distributed parallel system like our platform, both of them would involve an extremely heavy communication. They are good for a multiprocessor hardware dedicated for video signal processing, or they may be well-suited for a vectorcomputer.

The next level of parallel processing is offered by the slice layer. Slices are always encoded independently of each other. Our first algorithm is based on this fact. However, we will show that the computation/communication gets too low over a certain number of machines, although the load balancing remains good. A self-explaining idea would be to encode frames in parallel. The parallel Berkeley encoder is essentially based on this method, the machines encode a certain amount of images according to their power. It ensures an excellent load balancing, but the coding may be done incorrectly. Figure 6 shows the problem, assuming that pictures 0-6 are assigned to processor P1, while pictures 7-12 are assigned to processor P2. P1 can start encoding the first frame because it is an intra. However, P2 cannot encode frame 7 (B) because it uses its references (frame 6 and frame 9) at motion compensation. As we mentioned, the decoder reconstructs the P and B pictures by means of their reference(s) and motion vector field(s). However, only decoded references are available at the decoding side. As a consequence, the prediction should be done from decoded frames as well, otherwise an inequality occurs between the encoder and decoder. This results in a decoded P or B frame of worse quality. In addition, this error is propagated up to the next I picture because the consecutive P pictures are predicted from each other. Thus, in order to encode frame 7, we should have frame 6 and frame 9 in decoded form. Therefore, P2 should wait until P1 finishes with frame 6, which destroys the parallelism almost completely.

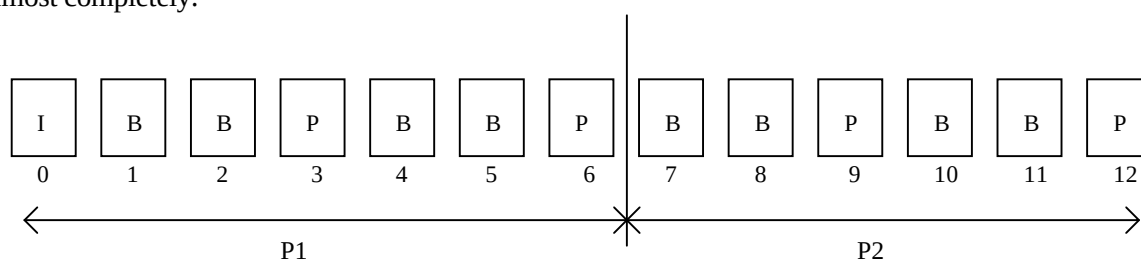


Figure 6. The problem of encoding frames in parallel

The current version of the parallel Berkeley encoder (1.5b) always uses original reference frames for motion compensation. The author claims that the degradation is negligible, and the use of a finer quantization can hide the differences. In our opinion a degradation of 2..3 dB is significant, and as we stated in the introduction, the production of high quality video is of our primary importance.

In fact, the motion estimation can be done independently of the other coding tasks, thus, P2 could find the vector fields for the P and B frames, and continue the processing when it is possible. Unfortunately, it would make the scheduling fairly complicated, and there is no guarantee for a proper synchronization. An other problem is that frame 6 is read by P1 and P2 as well, which increases the communication. However, none of the problems emerge if every run of frames starts with an I frame and ends with a P frame. This makes a group of frames independent from the others. In this way, the requirements for a Group Of Pictures are satisfied (see Section 2), therefore, we will call it GOP level parallelization. Our second algorithm is based on this method. Unfortunately, there are problems with this kind of parallelism as well. In order to ensure good quality at low bit-rate coding, the distance between two I frames should be large enough, typically 10...15. This means a large data size, which results in worse load balancing than the frame parallelization. We will show that using smaller GOPs like IBBP, not only provides good load balancing, but good parallel performance as well.

4. Description of the developed algorithms

4.1 The parallel model

In both cases the configuration is a master-slave model as shown in Figure 7. The slaves reside on different machines, there may be one slave on the master host. The master reads the raw images from a file, and it writes the compressed data (bitstream) into an other file. Since the system is prepared for MPEG-1 encoding, it is always required to transform the frames into 4:2:0 format. A hardware video digitizer usually outputs the signal in 4:2:2 format, and our image sequence files are also in this format. 4:2:2 means that every line contains chrominance samples, but the number of U and V samples is half of the number of Y samples per line. Consequently, a vertical chroma decimation is needed, which is done by the master process. Except for this one, all the other computation is performed by the slaves. The master controls the slaves, providing the load balancing in this way. It distributes raw image data with some control info among them. The slaves do the encoding, and send back the results to the master, either via the TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) sockets or via NFS. They do not communicate with each other.

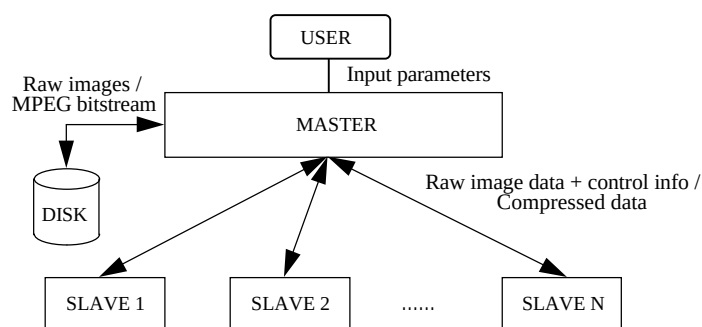


Figure 7. The parallel configuration

The user can contact only the master by entering several coding parameters, such as the GOP pattern, the motion vector range, the number of slaves to be started, the quality of the video, etc. In our case every GOP contains exactly one I picture. After some successful initialization steps the encoding begins. At the end of the processing the slaves provide the master with their computation+ communication and idle times in order to evaluate the performance of the system.

4.2 The slice level algorithm

In this case, the parallelization is done within a frame (spatial parallelization). The load balancing can be dynamic on GOP level, but it is always static within a GOP. We apply the following definitions according to Figure 8. A slice consists of any number of horizontal macroblock rows. A segment consists of a slice and the associated extra areas above and/or below the slice if this slice comes from an I or P picture. These extra areas are needed for motion estimation/compensation. Since B pictures are never used as reference, in those cases no extra areas are required.

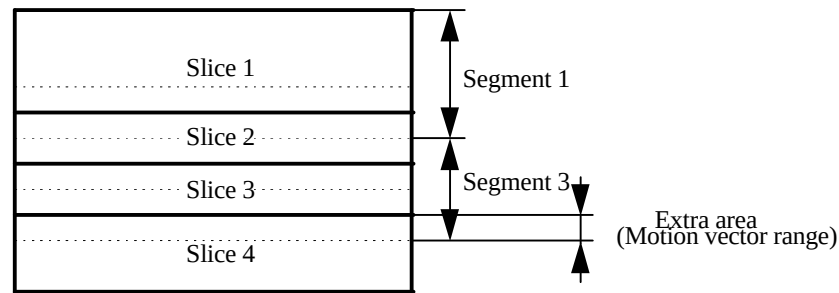


Figure 8. Illustration of slices and segments (4 slaves in the configuration)

A frame contains as many slices as the number of slaves in the configuration. This provides the least communication start-ups per frame. Segments are assigned to the slaves according to their instantaneous power. The faster a certain slave, the larger the slice it processes. The slaves' power is determined by a test procedure. The distribution of the workload is constant between two tests, a given slave processes the same amount of MB rows in the same position. A test can happen at any I frame. During a test the actual I picture is multicast among the slaves, and each of them encodes it entirely. The results are sent back accompanied by the encoding times. Using these times the master can estimate the slaves' performance. It should be noted that frequent tests improve the load balancing, but increase the computational load as well. In the normal mode of operation each slave sends back its encoded slice, which may not arrive in the right order. In such a situation the master stores them temporary, but it puts them in the MPEG file as soon as possible. The communication is always done via the TCP and UDP sockets, the machines do not have to be mounted to the NFS.

A consequence of this method is that some parts of the predicted pictures may not be encoded correctly. The extra areas on the reference frames should also be encoded and decoded in order to make the prediction correctly. They can be processed if the actual segment comes from an I frame, but they cannot if it derives from a P frame. Only the first P frame following an I frame can be encoded correctly sure enough. All the other predicted frames referenced by this P picture will contain slices encoded incorrectly if there are motion vectors pointing to the unprocessed extra areas. For instance, assume the following sequence: IBBPBBPIBBP... Except for the first P frame, a slice in any other predicted frames up to the next I picture cannot use decoded extra areas because they are present at other slaves. Of course a given slave could acquire these areas in decoded form, but it would introduce a heavy communication between the slaves. Incorrect encoding occurs only if there is a vertical movement close to a segmentation line. In addition the more slaves are in the configuration, the more extra areas we have. Fortunately, live video sequences always contain much more horizontal displacements than vertical ones, which prevents the video material from quality degradation in this case. However, we did a little test in order to examine the quality if a strong vertical movement happens in a scene. The details can be found in Section 5.

4.3 The GOP level algorithm

This method processes Groups Of Pictures in parallel (temporal parallelization). The scheduling is much simpler than in the previous case, even the load balancing is dynamic throughout the sequence. It works in a round-robin fashion as follows: Having N slaves, the first N GOPs are assigned to them in turn. Before encoding the last frame in a GOP, the slave signals a request towards the master for new work. If there are no more GOPs left, the master shuts down the just freed slave. After serving the slave the master prepares the next GOP for sending. While the raw GOPs are distributed via the UDP and TCP sockets, the encoded ones are put in separate files by the slaves. It is assumed that the machines in the configuration are mounted to the NFS. At the end of the processing the master tries to concatenate the encoded groups. If there is no NFS, the user should transfer the results to one place and combine them itself. The preparation work for a GOP (reading, UV decimating) can take a significant time, especially if the frames are read from the NFS. A slave request may arrive when the master works, thus, the caller cannot be served immediately. Since that slave is still working, it is likely to stay in idle state for a shorter period as if it signalled after finishing.

5. Tests and results

5.1 Performance test

Two kinds of standard measure, the speed-up and the efficiency were used to evaluate the performance of the parallel system with the presented algorithms. In our case they are defined as follows:

$$\text{speed-up} = \frac{\text{time for sequential encoding}}{\text{time for parallel encoding}} \quad \text{efficiency} = \frac{\text{speed-up for } M \text{ machines}}{M}$$

We used SUN Sparc 2s connected via Ethernet for doing experiments. Although the machines were identical, their load varied in time. Therefore, the encoding times (for sequential and parallel cases) also fluctuated in time. The presented results are averaged from several measurements. During the tests the master and one slave process were running on the same host, while the other slaves were on different hosts. This ensured a better utilization of the system as if all the slaves had been totally separated from the master. In every case, 200 SIF frames were encoded at a motion vector range of 32. Two kinds of GOP pattern were used: IBBP ($N=4$) and IBBPBBPBBP ($N=10$). The images were always read from the NFS. In the case of slice level parallelization the number of test frames varied between 1 and 4, depending on the load of the system. The reference sequential encoding rates are 0.2 frames per second ($N=4$) and 0.18 frames per second ($N=10$).

Figure 9 indicates the speed-up as a function of the number of machines for both algorithms if the IBBP pattern is used. As we can see, the speed-up goes in saturation quite quickly if slices are encoded in parallel. It turns out that this level of parallelism is too fine for such a distributed system with the applied machines and network. On the one hand, the low bandwidth of the network should be responsible for the performance degradation. Usually the transfer rate is not more than 100 Kbytes/s. On the other hand, the communication activity gets higher increasing the number of slaves, since this latter is equal to the number of communication start-ups per frame. Furthermore, the amount of data to be transferred is higher than it should be, because the extra areas are always overlapped on an I or P frame, thus, they are sent twice. The speed-up is 3.3 with 6 slaves, achieving an encoding rate of 0.66 frames per second. However, the GOP level parallelism shows an almost linear speed-up in the denoted #machines interval. The communication overhead is reduced significantly by initiating a transfer much less times. Moreover, all image data is sent only once. The resulted speed-up is 4.7 with 6 machines, corresponding to a rate of 0.94 frames per second. As it is shown in Figure 10, the system works much more efficiently in this case.

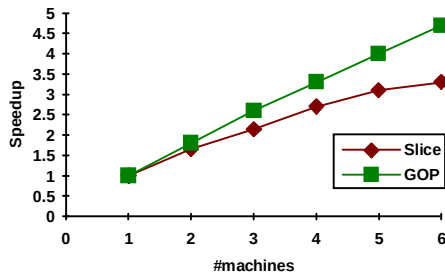


Figure 9. Speed-up vs. #machines (GOP=IBBP)

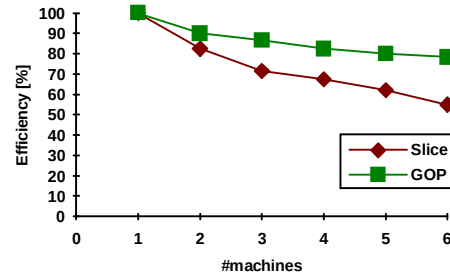
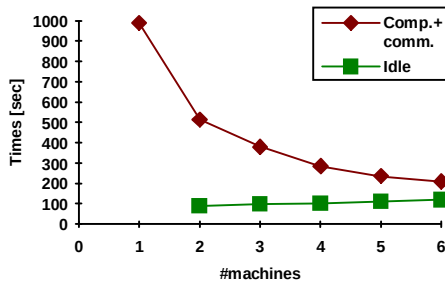
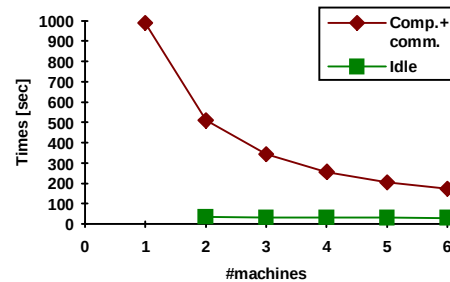


Figure 10. Efficiency vs. #machines (GOP=IBBP)

In order to estimate the computation/communication, Figure 11 shows the slaves' average computation+communication and idle times. The first two times are handled together because a process considers a transfer finished if data has been passed to the local daemon. A local daemon is running on each host, and its duty is to catch the messages coming to the processes on that host. As a consequence, this communication time is far less than the real one. The real communication time is characterized better by the slaves' idle time, assuming good scheduling (i.e. slight data contention only). As it is shown, the idle time gets comparable to the computation time if the slice parallelization is applied with 5 or 6 machines (Figure 11/a). The computation/communication is about 2 in these cases, that is extremely low. However, using the GOP parallelization, the average idle time is not only much smaller (by 60-70%), but it is almost constant in the examined #machines interval as well (Figure 11/b).



11/a Slice parallelization



11/b GOP parallelization

Figure 11. Average computation+communication and idle times of slaves vs. #machines (GOP=IBBP)

Despite the performance degradation of the slice-level algorithm, its load balancing control was working well, the differences in computation times were within 10%. Unfortunately, an exact segmentation is hardly possible because the number of MB rows is 18 (or 15) on a SIF image, and this was found too few in case of more than 3 or 4 machines. We experienced a better load balancing with larger sized images, but it could not hide the problem of intensive communication.

The next experiments were carried out at a larger GOP size ($N=10$). While a shorter pattern can be suitable for an application requiring fast random access (e.g. digital video recorder), a longer one may be appropriate for a low/middle bit-rate application (e.g. standard digital television). For fast comparison we denoted the previous speed-ups as well in Figure 12. The speed-up becomes slightly better at the slice parallelization because the number of predicted frames is higher, which involves more computation. However, it becomes slightly worse in the other case (Figure 13) because the load balancing degrades due to the larger data size. Of course encoding thousands of pictures will surely improve the performance because the relative differences in the computation times will be less. Unfortunately, it is expected that using plenty of machines, the performance will get worse also at the GOP level parallelization because the master will not be able to serve the slaves in time. The speed-up will go in saturation, and finally, it will fall down if the number of machines is increased unlimited.

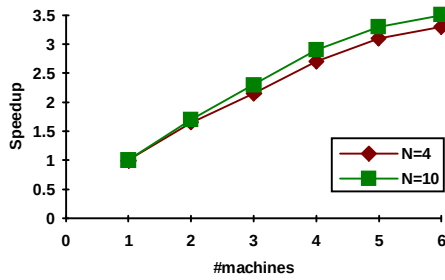


Figure 12. Speed-up vs. #machines (Slice level)

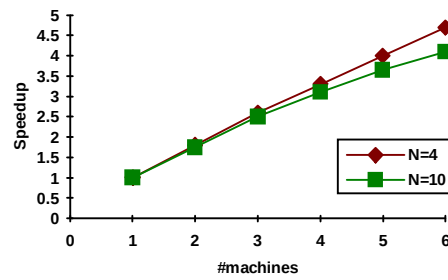


Figure 13. Speed-up vs. #machines (GOP level)

Finally, it is worth comparing the performance of the parallel Berkeley encoder with that of our one using the GOP level algorithm. Also, the parallel Berkeley implements the coding scheme on a master-slave model. The scheduling is based on the Pool of Tasks paradigm, a slave is kept busy by assigning more work at the beginning and less work at the end of the processing to it. It provides an excellent load balancing at the expense of incorrect encoding. Figure 14 shows the curves. The GOP pattern of IBBP was used for the comparison. All the other encoder settings were the same as before except for the motion estimation. Since the Berkeley encoder does not support hierarchical search methods, we used a full search with a maximum displacement of 8, providing approximately the same sequential reference time. As we can see, there is a very slight difference between the curves. Our algorithm works well preserving the quality of the video material.

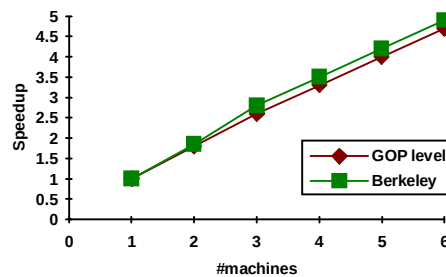


Figure 14. Speed-up vs. #machines (GOP level and Berkeley at GOP=IBBP)

5.2 Quality test for slice parallelization

As it was mentioned in Section 4, the slice parallelization introduces a problem. Some parts of the predicted frames are encoded incorrectly if vertical motion is present close to the segmentation borders. In order to examine a really bad case, we encoded a special test sequence containing heavy vertical movement. We used 6 slaves and the following GOP patterns: IBBP and IPPP. In the first set only the B pictures were encoded incorrectly, while in the second case an error propagation was introduced as well. The decoded video was evaluated using the Signal to Noise Ratio compared it to that of the correctly encoded signal. The degradation was within 0.05 dB at B pictures and less than 0.5 dB at P pictures. These extreme cases guarantee that the quality will be preserved in normal situations.

5.3 Conclusion of the tests

We can conclude the followings from the above tests: The slice parallelization provides a reasonable speed-up if no more than 3 or 4 machines are used. If the sequence contains only a few frames, and the use of larger GOPs is necessary, this algorithm is more optimal than the other one. In any other case the GOP parallelization is proposed, especially if the GOP consists of a few frames. If the use of larger GOPs is inevitable, the good load balancing can be ensured by encoding more frames.

6. Discussion

One of the main bounds for the system performance was the high acquisition time of the frames. In our experiments the images were always read from the network file system. The size of a frame in SIF is 150,000 Kbytes about. The original frames were larger by 30% because of the 4:2:2 format. We could not achieve a reading rate of higher than 2 frames per second, that means the network speed was 400 Kbytes/s in the best case! Consequently, we can have the fastest workstations in the world, we will not be able to achieve an encoding rate of higher than 2 frames per second. A conventional local disk does not solve the problem either, since it usually has an average transfer rate of 1 Mbyte/s. In order to get the frames from the media in real-time, it is essential to use multiple disks. Beside having a high-speed media, it is important to use a high-speed network like FDDI or ATM because transferring raw images means a high traffic load on the communication channel. The slice parallelization will probably work better on such a system.

At the slice parallelization the following possibility may be offered: every frame could be sent to each slave by a multicast command. A slave would get an additional control info on the number of MB rows to be processed as well as the position of the first row. In this way, the number of communication start-ups is reduced to 1 per frame. Moreover, there are no additional areas to be sent twice. We tried this method, but the system worked much worse. The average idle time at the slave side increased by 50-100%. The explanation might be the following: When multicasting a message, data is routed through the daemons. At every daemon it is copied out and passed to the corresponding process. The routing takes more time than sending the messages directly to all the processes. Our future work will be based on improving the GOP level parallelization further. This not only involves developing better scheduling methods, but it is intended to add a highly sophisticated bit-rate control algorithm as well.

7. Conclusion

In this paper we presented two kinds of parallel algorithm for the MPEG video compression by using the PVM library. Encoding slices in parallel introduced too much communication compared to the computation. The speed-up was 3.3 with 6 Sparc 2s on the Ethernet. The GOP level parallelization worked much better, achieving a speed-up of 4.7 with 6 machines. The algorithms are entirely portable, they only require an installed PVM version of 3.3.x. The current system produces an MPEG-1 video stream, but the methods can be applied in the MPEG-2 encoding as well.

This work was performed with the support of TEMPUS JEP 3815 and of the European Community through the COPERNICUS 1463 ATMIN project.

References

- [1] Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to 1.5 Mbit/s, ISO 11172 (MPEG-1 standard), 1993.
- [2] Generic Coding of Moving Pictures and Associated Audio, ISO/IEC 13818 (MPEG-2 standard), 1994.
- [3] C. Bouville et al.: "A Real-time Testbed for MPEG-1 Video Compression", *Proc. of the SPIE on Visual Communications and Image Processing*, Vol. 2094 (1993), pp. 205-212.
- [4] Al Geist et al.: "PVM: Parallel Virtual Machine - A User's Guide and Tutorial for Networked Parallel Computing", The MIT Press, 1994.
- [5] K. L. Gong: "Parallel MPEG-1 Video Encoding", MS Thesis, Dept. of EECS, University of California at Berkeley, May 1994, Issued as *Technical Report 811*.
- [6] K. Shen, L. A. Rowe, E. J. Delp: "A Parallel Implementation of an MPEG-1 Encoder: Faster Than Real-time!", *Proc. of the SPIE-The International Society for Optical Engineering*, Vol. 2419, pp. 407-418.
- [7] Moschytz et al.: "Practical Fast 1-D DCT Algorithms with 11 Multiplications", *Proc. Int'l. Conf. on Acoustics, Speech and Signal Processing 1989 (ICASSP '89)*, pp. 988-991.