A Parallel Implementation of H.26L Video Encoder

J. C. Fernández[1] and M. P. Malumbres[2]

[1] Dept. de Ingeniería y Ciencia de los Computadores, Universidad Jaume I, 12071-Castellón (Spain), Phone: +34-964-728265; Fax: +34-964-728435, e-mail: jfernand@inf.uji.es

[2] Dept. de Informática de Sistemas y Computadores, Universidad Politécnica de Valencia, 46071-Valencia (Spain), Phone: +34-96-3879703; Fax: +34-96-3877579, e-mail: mperez@gap.upv.es

Abstract. Over the last decade a lot of research and development efforts have gone into designing competitive video coding standards for several kinds of applications. Some video encoders like MPEG-4 and H.26L, exhibit a high computational cost, far from real-time encoding, with medium to high quality video sequences. So, for these kinds of video coding standards, it is very difficult to find software solutions able to code video in real-time. In this paper, we design a parallel version of the ITU-T H.26L video coding standard, showing different implementation approaches and evaluate their performance. Several experiments were carried out showing that parallel versions of H.26L significantly improve its coding speed by running in a cluster of personal computers interconnected with Myrinet LAN.

1 Introduction

The storage, processing and delivery of multimedia data in their raw form is very expensive; for example, a standard 35mm photograph could require about 18 MBytes of storage and one second of NTSC-quality colour video requires almost 23 MBytes of storage. To make widespread use of digital imagery practical, some form of data compression must be used.

Digital images can be compressed by eliminating redundant information. There are three types of redundancy that can be exploited by image compression systems:

– Spatial Redundancy. In almost all natural images, the values of neighbouring pixels are strongly correlated.
– Spectral Redundancy. In images composed of more than one spectral band, the spectral values for the same pixel location are often correlated.
– Temporal Redundancy. Adjacent frames in a video sequence often show very little change.

The removal of spatial and spectral redundancies is often accomplished by transform coding, which uses a reversible linear transform to decorrelate the image data. Temporal redundancy is exploited by techniques that only encode the differences between adjacent frames in the image sequence, such as motion prediction and compensation.

In the last few years, many video compression algorithms have been proposed. As a result, several image and video compression standards have been approved [H.26X, MPEG-X, JPEG2000] and many hardware/software solutions are now available. Futhermore, clusters of workstations (COWs) are currently being considered as a cost-effective alternative for small-scale parallel computing. Although COWs do not provide the computing power available in multicomputers and multiprocessors, they meet the needs of a great variety of parallel computing problems at a lower cost, in particular high quality video coding applications for video on demand systems.

With respect to video parallel programming, several MPEG parallel implementations have been developed. Perhaps, at software level, the most significant works on MPEG parallelization have been done at Berkeley [6], [18]. Load balancing studies have been performed at Purdue, achieving significant improvements on a Paragon machine [15]. Yu and Anastassiou have evaluated parallel versions of MPEG-2 encoder on an Ethernet-based COWs [16]. Other MPEG parallel implementations have been proposed to run on high performance switched LAN networks [10]. Also, several H.263 parallel implementations have been developed on multiprocessors [17] and COWs [4].

In this paper, a parallel ITU-T H.26L video encoder is proposed. The computational cost of the H.26L encoder is extremely high. We propose two parallel versions: the first divides the overall video sequence among the working nodes (GOP-level parallelism). The second divides one frame among the working nodes (Frame-level parallelism). Both approaches were evaluated on a Myrinet-based COWs.

In section (2) we give a brief description of the H.26L encoder. In section (3) the two parallel versions are explained. Section (4) shows some evaluation results and, finally, in section (5) some conclusions are drawn.

2 The H.26L Video Encoder

H.26L [8] is the current project of the ITU-T Video Coding Experts Group (VCEG), a group officially chartered as ITU-T Study Group 16 Question 6. The primary goals of the H.26L project are:

– Improved coding efficiency. The syntax of H.26L should permit an average reduction in bit rate by 50% compared to H.263+ (version 2 of H.263 [9]) for a similar degree of encoder optimization.
– Improved Network Adaptation. Issues relating to network adaptation that were examined seriously for the first time in the H.263 and MPEG-4 projects are being taken further in H.26L.
– Simple syntax specification. The design of H.26L is strongly intended to lead to a simple, clean solution avoiding any excessive quantity of optional features or profile configurations. A new feature of the design is its introduction

of a conceptual separation between a Video Coding Layer (VCL), which provides the core high-compression representation of the video picture content, and a Network Adaptation Layer (NAL), which packages that representation for delivery over a particular type of network.

Different from the previous MPEG and ITU-T standards, some new techniques, such as spatial prediction in intra coding, motion compensation with adaptive block size, 4x4 integer DCT, UVLC (Universal Variable Length Coding), CABAC (Context-based Adaptive Binary Arithmetic Coding) and loop filter are adopted by H.26L. The intra predictions are derived from the neighbouring pixels in left and top blocks. The unit size of spatial prediction is either 4x4 or 16x16. There are 6 different modes for each 4x4 sub-block. The first mode is DC prediction and other modes represent different directions of predictions. 16x16 intra prediction is particularly suitable for a flat region with little details. Vertical, horizontal, DC and plane predictions are used at 16x16 size.

H.26L allows more than one previous frames for inter frame prediction. Inter prediction is calculated from one of these previous frames. In the MPEG-4 standard, only 8x8 and 16x16 blocks are the units for motion estimation and compensation. However, seven block sizes, i.e., 16x16, 16x8, 8x16, 8x8, 8x4, 4x8 and 4x4, are supported in H.26L. The spiral search finds the minimum cost for each block size in the given range. The cost includes signal SAD and overhead bits for coding block size information and motion vectors. The optimal block size is decided based on these minimum costs. If 4x4 block size is the winner, there are 8 motion vectors for this macroblock. The precision of motion vectors is at least quarter pixel. With higher complexity for higher coding efficiency, H.26L allows 1/8 pixel accuracy prediction. The residue after prediction is transformed with 4x4 integer DCT. Basic scanning order is still zigzag similar to that used in MPEG-4. Two different entropy-coding techniques are used in H.26L to compress quantized coefficients: UVLC and CABAC. UVLC provides a simple and robust method to code all mode information and DCT coefficients. But the performance at moderate or high bit rates is not good. Therefore, CABAC is proposed as another option in H.26L.

The sum of the prediction and the current reconstructed error image forms the reconstructed reference. H.26L uses the deblocking filter in the motion compensation loop. The deblocking process directly operates on the reconstructed reference first across vertical edges and then across horizontal edges. Obviously, different image regions and different bit rates need different levels of smoothing. Therefore, the deblocking filter is automatically adjusted in H.26L depending on activities of blocks and QP parameters.

3 Parallel Algorithms

The amount of work associated with coding different pictures is variable and unpredictable. In this section, we present two versions of an H.26L parallel video encoder: GOP (Group Of Pictures) division and frame division. The difference between them consists of the degree of parallelism employed. The first version, GOP division, divides the overall video sequence among the available working nodes, so each node is able to independently process a set of contiguous frames. With GOP division there is no explicit communication between working nodes at encoding time. The only communication is performed at the end of the encoding process when results have to be delivered to the final compressed file. The second parallel version of the H.26L video encoder, frame division, divides one frame among working nodes (fine-grain parallelism). The considered task unit for frame division is the slice, a consecutive group of macroblocks. As expected, when processing a frame two communications steps are performed: (a) storing the compressed bitstream associated to this frame (gather operation) and, (b) exchange state information required for coding the next frame.

3.1 GOP division

As stated above, each processor computes a GOP of the video sequence. Each GOP begins with an I-Frame, the rest being P-Frames and, optionally, B-Frames. So, if the first picture is an I-Frame, that does not depend on previous pictures, then a GOP is defined as a closed group of pictures that can be decoded independently. Let us consider the following values, n_frames is the number of frames in the video sequence, n_frames_gop is the number of frames in one GOP, n_gops (the number of GOPs) is given by n_frames/n_frames_gop and p is the number of processors. The number of not assigned GOPs is $n\_gops\_not = mod(n\_gops, p)$ and the number of assigned GOPs to each processor is $n\_gops\_as = (n\_gops - n\_gops\_not)/p$. The total number of GOPs assigned to the processor P_k, $k = 0 : p-1$, is given by:

$$n\_gops\_p = \begin{cases} n\_gops\_as + 1, \; k < n\_gops\_not \\ \\ n\_gops\_as, \; k \geq n\_gops\_not \end{cases} \tag{1}$$

To determine which frames will be assigned to each processor, two parameters have been defined, if_k, the initial frame, and ff_k the final frame belonging to processor P_k. Then P_k calculates the frames $if_k, if_{k+1}, \cdots, ff_k$. The values of these parameters are given by the following expressions:

$$if_k = \begin{cases} k * (n\_gops\_p * n\_frames\_gop), \; k < n\_gops\_not \\ \\ n\_frames - ((p-k) * (n\_gops\_p * n\_frames\_gop)), \; k \geq n\_gops\_not \end{cases} \tag{2}$$

$$ff_k = \begin{cases} ((k+1) * (n\_gops\_p * n\_frames\_gop)) - 1, \; k < n\_gops\_not \\ \\ (n\_frames - ((p-k-1) * (n\_gops\_p * n\_frames\_gop))) - 1, \; k \geq n\_gops\_not \end{cases} \tag{3}$$

Thus, the parallel algorithm using the GOP division method will be the following:

```
**Compute_GOP_par**: In Parallel for $k = 0, \cdots, p-1$
In Processor $P_k$
    (*Compute frames *)
    Compute $if_k$ and $ff_k$
    For $i = if_k$ to $ff_k$ do
      Compute Frame $i$
    EndFor
    (*Obtain bitstream*)
    Send bitstream to $P_0$
    If $k = 0$ obtain final compressed file
End Compute_GOP_par
```

3.2 Frame division

In this case the task unit is the slice, a group of consecutive macroblocks in a frame. Each frame is divided in slices, and those are assigned to working nodes. Since most test sequences use only one slice per row of macroblocks (the slice width is the same as the frame width) , each frame usually contains a small number of slices. For example, in CIF video format the number of macroblocks that conforms a slice will be 22, with a total of 18 slices per frame. With QCIF formats a slice will contain 11 macroblocks, so the total number of slices is limited to 9.

Let us consider the following values, n_frames is the number of frames in the overall sequence, n_m_t is the number of macroblocks in one frame, n_m_s is the number of macroblocks in one slice, $n\_s = n\_m\_t/n\_m\_s$ is the number of slices and p is the number of processors. The number of not assigned slices is $n\_s\_n = mod(n\_s, p)$ and the number of assigned slices to each processor is $n\_s\_as = (n\_s - n\_s\_n)/p$. The total number of slices assigned to the processor P_k, $k = 0 : p-1$, is given by:

$$n\_s\_p = \begin{cases} n\_s\_as + 1, \;\; k < n\_s\_n \\ \\ n\_s\_as, \;\; k \geq n\_s\_n \end{cases} \tag{4}$$

To determine the macroblocks assigned to each processor, two parameters have been defined, im_k, the initial macroblock, and fm_k the final macroblock of processor P_k. Then P_k calculates the macroblocks $im_k, im_{k+1}, \cdots, fm_k$. The values of these parameters are given by the following expressions:

$$im_k = \begin{cases} k * (n\_s\_p * n\_m\_s), \;\; k < n\_s\_n \\ \\ n\_m\_t - ((p-k) * (n\_s\_p * n\_m\_s)), \;\; k \geq n\_s\_n \end{cases} \tag{5}$$

$$fm_k = \begin{cases} ((k+1) * (n\_s\_p * n\_m\_s)) - 1, \ k < n\_s\_n \\ \\ (n\_m\_t - ((p-k-1) * (n\_s\_p * n\_m\_s))) - 1, \ k \geq n\_s\_n \end{cases} \tag{6}$$

Thus, the parallel algorithm using the frame division method will be the following:

```
Compute_frame_par: In Parallel for $k = 0, \cdots, p-1$
In Processor $P_k$
  (*Compute macroblocks *)
  Compute $im_k$ and $fm_k$
  For $i = 0$ to $n\_frames - 1$ do
    For $j = im_k$ to $fm_k$ do
      Compute Macroblock $j$ of Frame $i$
    EndFor
    Send bitstream to $P_0$
    If $k = 0$ obtain partial bitstream
  EndFor
  If $k = 0$ obtain final compressed file
End Compute_frame_par
```

4 Experimental Results

The sequential algorithm is evaluated using the sequential execution time, T_s. The parallel algorithms are evaluated using parallel execution time T_p (p processors), speed-up, $S_p = T_1/T_p$ and efficiency, $E_p = S_p/p$. The results have been obtained using a Beowulf cluster of 32 nodes interconnected with a Myrinet switch. Each node consists of an Intel Pentium-II processor at 300MHz with 128 MBytes RAM. Communication routines in MPI and C language are used. Several numbers of working nodes have been used for running both parallel versions of H.26L encoder, in particular we have run experiments with 2, 3, 4, 9, 16 and 24 nodes. We have used the public available sources of H.26L TML 8.4 as the starting point for this study. During the experiments several QCIF (176x144) video sequences (carphone, claire and miss-am) of 315 frames were used. The sequence of frame types used in all tests is the following $IBBPBBPI\cdots$. In the GOP division method $n\_frames\_gop = 7$ and $n\_gops = 45$. Nine slices of 11 macroblocks have been considered. In the frame division method $n\_m\_t = 99$, $n\_m\_s = 11$ and $n\_s = 11$. The deblocking filter optional mode was not used. We have employed the following coding options:

- Fractional 1/4 pixel precision is used.
- Seven different block sizes ($16 \times 16, 16 \times 8, 8 \times 16, 8 \times 8, 8 \times 4, 4 \times 8$ and 4×4) are employed in motion compensated prediction.
- One previous frame is used for inter motion search.

The operations of reading from the source video file and writing into the output compressed file are included in the total encoding time. In order to verify the correctness of parallel implementations, the resulting compressed video streams were checked in terms of quality metrics and total number of bits per frame, being exactly the same results as those obtained with the sequential version.

Tables 1, 2 and 3 show the experimental results in seconds for the first 45 GOPs of each video sequence.

Table 1. Experimental results using the carphone video sequence.

GOP division				Frame division			
p	T_p	Speed-up	Efficiency	p	T_p	Speed-up	Efficiency
1	9365.362			1	9365.362		
2	4714.796	1.986	99.3%	2	5160.32	1.81	90.74%
3	3233.557	2.896	96.54%	3	3369.01	2.77	92.66%
4	2468.485	3.794	94.84%	4	3359.01	2.78	69.68%
9	1099.195	8.52	94.66%	9	1324.061	7.07	78.59%
16	620.031	15.10	94.40%				
24	425.395	22.01	91.73%				

Table 2. Experimental results using the claire video sequence.

GOP division				Frame division			
p	T_p	Speed-up	Efficiency	p	T_p	Speed-up	Efficiency
1	9448.893			1	9448.893		
2	4723.29	1.98	99.18%	2	5231.495	1.80	90.30%
3	3129.467	2.99	99.54%	3	3347.598	2.82	94.08%
4	2486.894	3.799	94.98%	4	3299.278	2.86	71.59%
9	1052.625	8.97	99.73%	9	1296.904	7.28	80.95%
16	638.019	14.809	92.56%				
24	426.059	22.17	92.40%				

As can be seen, the GOP division method efficiencies are very good. The distribution of GOPs between working nodes was the following:

- 2 nodes: 23 GOPs for first node and 22 for the second one.
- 3 nodes: 15 GOPs for each node.
- 4 nodes: 12 GOPs for the first node and 11 for the remaining nodes.
- 9 nodes: 5 GOPs per node.
- 16 nodes: 3 GOPs for nodes P_k, $k = 0 : 12$, and 2 GOPs for the remaining nodes.

Table 3. Experimental results using the miss-am video sequence.

GOP division				Frame division			
p	T_p	Speed-up	Efficiency	p	T_p	Speed-up	Efficiency
1	11948.259			1	11948.259		
2	6404.774	1.86	93.27%	2	6669.15	1.79	89.57%
3	4367.796	2.73	91.18%	3	4277.01	279	93.12%
4	3213.745	3.71	92.94%	4	4145.189	2.88	72.06%
9	1463.771	8.162	90.69%	9	1622.632	7.36	81.81%
16	876.57	13.63	85.19%				
24	585.900	20.39	84.97%				

– 24 nodes: 2 GOPs for nodes P_k, $k = 0 : 20$, and 1 GOP for the remaining nodes.

When using the frame division version of H.26L video encoder, the best results, in terms of efficiencies, are obtained using three processors. This is mainly due to good load balance, assigning three slices (33 macroblocks) to each node. When other numbers of nodes are employed, the frame division will be:

– 2 nodes: 5 and 4 slices per node.
– 4 nodes: 3 slices for the first node and 2 for the rest of nodes.
– 9 nodes: 1 slice per node. Here the load balancing is the same as that obtained with 3 nodes, however, communication time significantly increases, reducing the total encoding time to 80%.

To compare both parallel versions the following figures are presented. Figure 1 shows the encoding time for the carphone video sequence.

In general the behaviour of frame division, in terms of encoding time, is worse than GOP division, because the former requires more communications than the latter.

Figure 2 shows the efficiencies of both parallel versions encoding the carphone video sequence.

The best efficiencies are obtained with GOP division, which achieves a good load balance. In the frame division there is one communication per frame, but in the GOP division there is only one communication during the total encoding time.

5 Conclusions

We have presented a preliminary study of two parallel implementations of a H.26L video encoder. The first method distributes the whole video sequence between system nodes, dividing the original sequence in Groups Of Pictures (GOPs). The second method proposed, divides each video frame among the

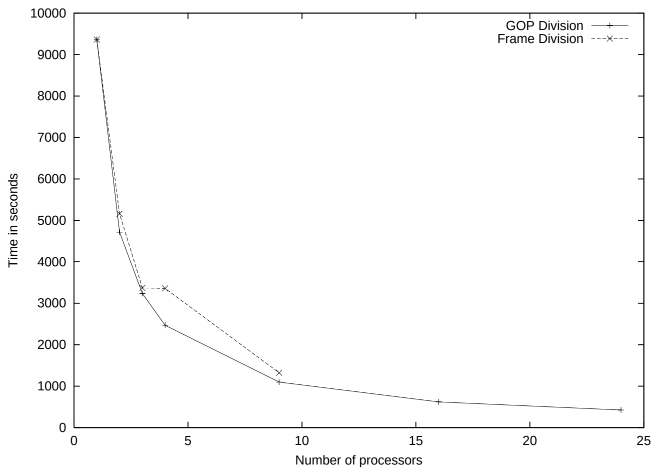

Fig. 1. Encoding time of the carphone video sequence

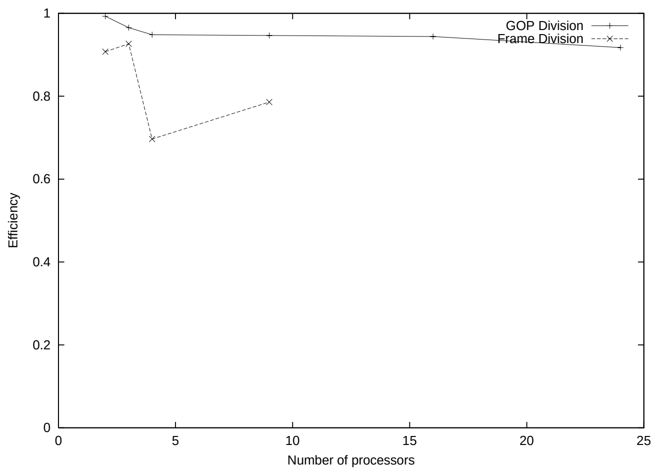

Fig. 2. Efficiencies of the carphone video sequence

nodes. The division was made at slice level, requiring a barrier synchronization between working nodes in order to exchange the necessary information to properly code the next frame. We have used common test sequences to evaluate the performance of both approaches, achieving better results with the GOP division method, because it requires only one communication step.

As future work, we are planning to make several optimizations to the frame division method in order to hide its communication overhead. Also, we will test other data distribution strategies to minimize communications in both methods.

References

1. Akramullah, S.M., Ahmad, I., Liou, M.L.: Parallelization of MPEG-2 Video Encoder for Parallel and Distributed Computing System. Midwest Symposium on Circuits and Systems, vol. 2. Rio Janeiro, Brasil (1995) 834-837
2. Akramullah, S.M., Ahmad, I., Liou, M.L.: Portable and Scalable MPEG-2 Video Encoder on Parallel and Data Distributed Computing Systems. Symposium on Visual Communications and Image Processing, vol. 27. Orlando- Florida (1996) 973-984
3. Akramullah, S.M., Ahmad, I., Liou, M.L.: Performance of a Software-Based MPEG-2 Video Encoder on Parallel and Data Distributed Computing Systems. IEEE Transactions on Circuits and Systems for Video Technology, vol. 7. (1997) 687-695
4. Akramullah, S.M., Ahmad, I., Liou, M.L.: A Software Based H.263 Video Encoder Using a Cluster of Workstations. Optical Science Engineering and Instrumentation Symposium, vol. 3166, San Diego- California (1997)
5. Cosmas, J., Paker, Y.P., Pearmain, A.J.: Parallel H.263 Video Encoder in Normal Coding Mode. IEEE Electronics Letters, vol. 34, no. 22 (1998) 2109-2110
6. Gong, K.L.: Parallel MPEG-1 video encoding. MS thesis, Department of EECS, University of California at Berkeley, Issued as Technical Report, May (1994)
7. Gove, R.J.: The MVP: a Highly-integrated Video Compression Chip. Proceedings of IEEE Data Compression Conference, 28-31, Utah, March (1994) 215-224
8. ITU-T: H.26L Test Model Long Project (TML-8), July (2001)
9. ITU-T: H.263 version 2, Video Coding for Very Low Bit Rate Communication (1998)
10. Kralevich, N.: The NOW Split-C MPEG Encoder. Internal Report Final Project CS267, University of California at Berkeley. Issued as Technical Report.
11. Long, K.L., Rowe, L.A.: Parallel MPEG-1 Video Encoder. Picture Coding Symposium, California (1994)
12. Martínez, L.F.: An Efficient ISO/MPEG-1 Layer II Encoder using Parallel Processing Techniques. Proyecto de investigación de la Universidad de Miami, Florida (1995)
13. Olivares, T., Quiles, F., Cuenca, P., Orozco-Barbosa, L., Ahmad, I.: Study of Data Distribution Techniques for the Implementation of an MPEG-2 Video Encoder. Proceedings of the IASTED International Conference Parallel and Distributed Computing and Systems, MIT Boston (1999)
14. Pacheco, P.S.: Parallel Programming with MPI, Morgan Kaufman Publishers, Inc.
15. Shen, K., Rowe, L.A., Delp, E.J.: A Parallel Implementation of an MPEG-1 Encoder: Faster then Real Time!. Conference on Digital Video Compression on Personal Computers: Algorithms and Technologies, San Jose- California (1995) 407-418.
16. Yu, Y., Anastassiou, D.: Software Implementation of MPEG2 Video Encoding using Socket Programming in LAN. Proceedings of the SPIE conference of Digital Video Compression on Personal Computer: Algorithm and Technology 7-8. San Jos-California, Feb. (1994) 229-240
17. Zheng, L., Cosmas, J., Itagaki, T.: Real Time H.263 Video Encoder Using Mercury Multiprocessor Workstation.
18. ftp://mm-ftp.cs.berkeley.edu/pub/multimedia/mpeg/encode

This article was processed using the LaTeX macro package with LLNCS style