Performance evaluation of parallel MPEG-4 video coding algorithms on clusters of workstations

A. Rodriguez, A. González and M.P. Malumbres
Technical University of Valencia
Camino de Vera 17, 46071 Valencia, SPAIN
abrodleo@doctor.upv.es, {agt,mperez}@disca.upv.es

Abstract

During the last decade a lot of research and development efforts have been made to design competitive video codecs for several kinds of multimedia applications. Some video encoders like MPEG-4 and H.264 exhibit a very high computational cost, particularly in the case of high quality video sequences. Then, it is very difficult to find software solutions that efficiently code high-quality video in real-time or faster. We propose and evaluate several parallel implementations of the MPEG-4 standard encoder on clusters of workstations. We have performed extensive experiments showing that coding speed can be improved to more than double real-time requirements, being coding efficiency the same than the one in the sequential version.

1. Introduction

Nowadays there are several standards for video compression like ITU H.26x and MPEG [1][2][3]. Among them, MPEG-4 seems to have the most promising future. MPEG-4 video encoders are very effective reducing the size of the video stream, but the processing demand is very high for high quality video sequences. Although there are hardware MPEG-4 video encoders available, they have severe restrictions (resolution, coding options, etc). A more flexible choice is to use parallel implementations running on clusters [4].

To take advantage of the potential processing power of clusters of workstations, we use parallel programming techniques based on message passing. We have used MPI [5] because there are free implementations available and it is a widely accepted standard.

There are several approaches to obtain parallel versions of sequential video encoders. One of them is based on exploiting functional parallelism [6] by using the MPEG-4 object coding feature. This solution is limited by the inter-dependencies between video objects and by the number of available objects.

Another approach often used is the space-temporal parallelism [7] that consists on frame segmentation of every video frame into slices which are independently coded [8]. This method has good scalability, but it limited by slice synchronization overhead.

A more interesting solution is to decompose the video sequence into GOPs (Groups Of Pictures) [10], then every GOP is independently processed by a dedicated processor. This basic scheme can be refined to achieve near linear speed-up [9]. In this work we work around this approach with an MPEG-4 encoder to reach or overpass real time encoding of high-quality video.

We focus our attention on the data distribution and GOP allocation mechanisms to get a balanced load and to reduce the impact of communication overhead. Also the resulting video quality is considered, in order to preserve the quality of the reconstructed video in our parallel versions.

This paper is organized as follows: In section 2 we show the data distribution process and the GOP allocation schemes. Section 3 describes the mechanism employed to hide the communication overhead. In section 4 we show performance evaluation results of the parallel MPEG-4 encoders implemented. Finally, in section 5 we summarize some conclusions and future work.

2. Parallel data distribution.

The basic idea for data distribution is to arrange the uncompressed video sequence in GOPs. Then, we have to decide (a) how processors get the GOPs, and (b) which GOPs correspond to each processor.

In order to determine the access to GOPs, we have considered two alternatives: (a) processor “0” accesses to disk and sends GOPs to the corresponding processors by means of MPI messages, and (b) every processor gets his GOPs accessing to disk by means of NFS (Network File System) service.

We have evaluated both approaches on a basic algorithm showing that they offer similar performance results. Then, we decided to use the NFS access instead of MPI because the code becomes simpler.

In relation to load balance we found that the first parallel implementations did not face this aspect, showing noticeable unbalanced processor utilizations. Then we analyze two different GOP scheduling schemes: (a) access on demand with decentralized sequential distribution, and (b) access on demand supervised by processor 0

In the first approach (version A), every processor knows in advance which GOPS it has to encode. The wait time in the basic version is now avoided because processors can access the next pre-assigned GOP immediately after processing the previous one. With this strategy every processor encodes the same number of GOPs. This may introduce some imbalance because the GOP encoding time has a considerable variability particularly due to the variable motion degree and picture detail level.

In the second approach (version B), processor 0 dynamically assigns GOPs to other processors. This dynamic assignment improves load balance because after finishing one GOP another one is ordered. Then, processors that encode low-complexity GOPs will encode more GOPs than processors that encode high-complexity ones.

3. Hiding communication overhead

Until now, one processor waits until a complete GOP is received before it begins to encode. This does not represent a performance penalty if the communication time required to transfer a GOP is much less than the time required to process it, and coding delay is not a critical performance metric

In order to solve this potential performance leak, we modify the parallel algorithm to start encoding as soon as possible, in order to overlap GOP encoding and GOP reading processes. So, communication delays are hidden and do not contribute to the parallel execution time. The encoder allows this operation because it requires, in general, only one frame ahead to process the actual one.

To introduce this enhancement to our parallel algorithm we will employ a multithreading approach. Communication is managed by one thread that reads frames from disk while the actual frame is coded by another thread.

This approach is really effective if cluster nodes have more than one processor and support symmetric multiprocessing. Another option is to use MPI non-blocking communication primitives, but the effectiveness of this approach depends on the MPI implementation.

4. Experimental results

All the experiments have been performed on a cluster of PCs available at our University. The cluster configuration is the following:

- 36 available nodes.
- Each node has 2 Pentium III processors @ 1 Ghz with 512 MB RAM.
- 18 GB SCSI hard discs, all available via NFS.
- Suse Linux 7.3 operating system with Kernel 2.4.17
- Al nodes are interconnected through two non-blocking Fast Ethernet switches.

The codec we have used is Microsoft Fdam 2.3-001213. It is implemented in C++ and the parallel versions have been developed using GNU gcc compiler. The source video sequence we have used in all the tests is "mother_and_daugther" (QCIF format, 4:2:0 subsampling, at 10 fps). It is 300 frames long. The GOP size is 15 frames (IBBPBBPBBPBBPBB). In order to simulate longer video sequences, the original one has been cycled. All the tests have been performed with the cluster in exclusive mode, i.e. there was no other user load.

We measured the processor utilizations of the two schemes proposed above. As shown in Figure 1, the load balance with 64 GOPs was good up to 16 processors. However, when using 24 and 32 processors, both algorithms start to lose efficiency. This is due to the reduction in the number of GOPs per processor as the number of processors increases. Then the probability of load imbalance is higher. As it can be seen in Figure 1, the dynamic GOP scheduling (B) behaves much better than a pre-assigned GOP scheme (A).

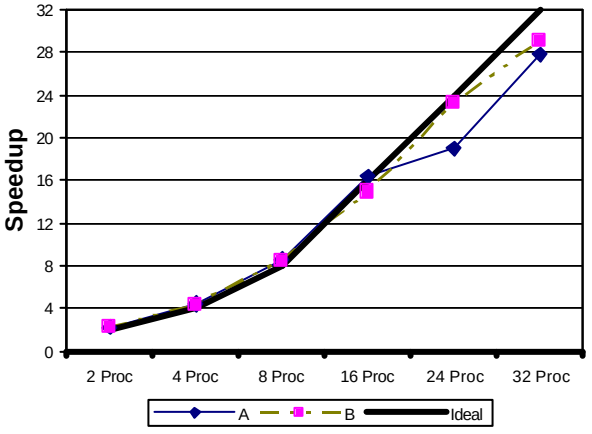

Figure 1: Speed-up for demand access schemes with 64 GOPs video length (sequential time is 426 seconds)

Now we consider hiding communication delays. This was solved with the multithreading approach (version Bm). The tests were performed using the same number of GOPS per processor instead of using the same stream length. As it can be seen in Figure 2, the improvement of the multithread approach is not very noticeable. This is because the communication time is very much lower than the computation time in this machine; remember we are using a Fast Ethernet interconnection network.

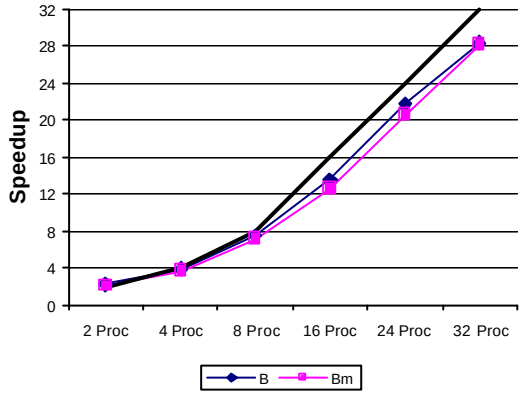

Figure 2: Effectiveness of overlapping communications and computation, with 2 Gops per processor

Also, we have run simulations with higher number of GOPs per processor, showing that the multithreaded version scales well, being equivalent to the original version (version B in figure 2) in terms of speedup.

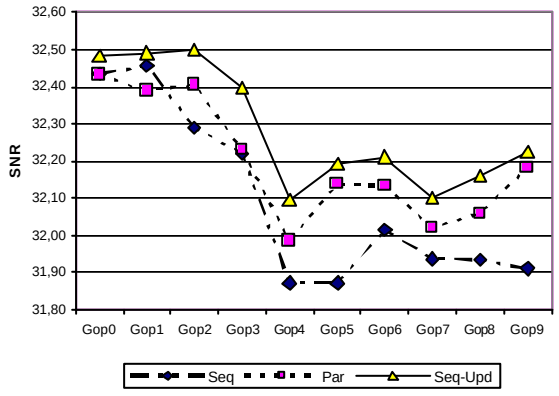

Figure 3. Comparing coding quality of sequential and parallel algorithms.

Finally, we have to consider the quality of the reconstructed video stream. A standard quality measure is the SNR (Signal-to-Noise Ratio). When comparing the SNR of reconstructed streams of both encoders the sequential (Seq), and the parallel version B (Par), we found differences in less than 1%, in favor to our parallel encoder (see Figure 3). This was due to the different GOP sequence used in both versions. Sequential version does not follow the same frame sequence in every GOP IBBPBBPBBPBBPBB-PBBPBB…, so there is only one Intra Frame (Frame I) in the whole sequence. However, our parallel version was fixed to use GOPs with the following frame sequence: IBBPBBPBBPBBPBB-IBBP… so each GOP can be independently processed by each node, with no data dependencies. We change the GOP sequence of the sequential algorithm (Seq-Upd) to the one used in our parallel version and again some differences appear. In this case, the sequential version gets better results in SNR (see Figure 3). This lead us to carefully verify the correctness of the coding process in order to find the cause of these differences.

After doing some tracing we noticed that our parallel version was no able to exactly follow the specified GOP sequence, due to the absence of first I frame of next GOP. So, the real frame sequence of our parallel version was IBBPBBPBBPBBPBB**P**, being the last frame of every GOP coded as P frame, instead of B frame.

To get the same coding results (quality and bitrate) than the sequential coder, we have modified the parallel version in order to code one extra frame per GOP: The first frame of the next GOP. This extra frame will be coded as an INTRA frame (I), to allow the correctly encoding of the two last B frames of current GOP. After introducing the changes in the code, the results of sequential and parallel version were exactly the same in terms of SNR.

For this improvement we have to pay an extra computational overhead that results, in average, less than 1% of the GOP computation time.

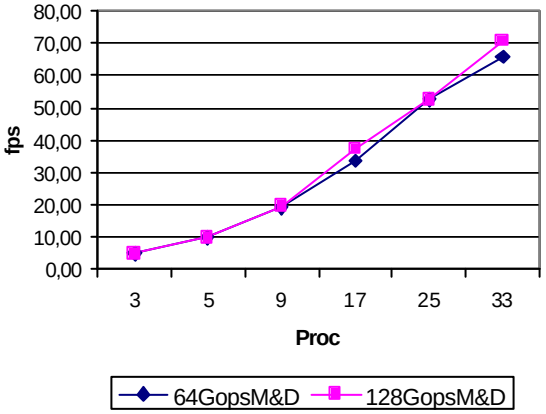

Figure 4. Coding speed of MPEG-4 parallel version for video sequences of 64 and 128 GOPs.

Finally Figure 4 shows the coding speed of our latest version (Bm). We have run the parallel encoder with two

versions of mother&daughter video sequence of different length (64 and 128 GOPs). The results show that our last parallel version is able to code faster than real-time from 5 processors and on (notice that the original video sequence was recorded at 10 frames per second). Also, with 33 processors we can reach encoding speeds up to 70 fps, which it is seven times faster than real-time (the original video frame rate). So, we can conclude saying that our parallel algorithm scales well on the target system in terms of encoding speed.

5. Conclusions and future work

We have tested several parallel encoders implemented from an open source MPEG-4 sequential encoder with the final goal of getting real-time and good quality video compression. There is a long way to reach this target and we have began by studying some important issues like how to access to the input data, how to get a data distribution without synchronization waits and with good load balance, how to avoid the impact of communication delays on execution time. At the same time the correctness of the implementation was proven in terms of video quality and compression bitrates. We have concluded that a simple parallel algorithm based on decomposing the input video in blocks that are processed independently gives good results.

However, our algorithms do not consider the encoding delay, what will be of special interest for encoding live video. That question will be part of future work along with a detailed study of the unbalanced nature of video sources that will identify and properly share the high complexity GOPs in high resolution video sequences. Also, we plan to combine several parallelism levels (SIMD, multithreading and message passing) in order to significatively reduce the parallel encoder execution time.

6. References

[1] Rob Koenen, "*International Organisation For Standardisation ISO/Iec JTC1/Sc29/Wg11 coding of Moving Pictures And Audio*"; http://www.cselt.it/mpeg/standards/mpeg-4/mpeg-4.htm, Marzo del 2001.

[2] J. Oliver, M.P. Malumbres. "*Compresión de Imagen y Video: Fundamentos Teóricos y aspectos prácticos*". Ed. UPV. 2001

[3] W. Effelsberg, R. Steinmetz. "*Video Compression Techniques*". Ed. Dpunkt-Verl. 1998.

[4] T. Olivares Montes, "*Codificación de Video MEPG-2 sobre una red de estaciones de trabajo*". Dpto. de Informática, Universidad de Castilla-La Mancha.

[5] P. Pacheco, "*Parallel programming with MPI*". Morgan Kauffman, 1997.

[6] A. Hamosfakidis,Y. Paker, *"Concurrency Analysis for Real Time MPEG4 Video Encoding"*,ICMCS, Vol. 2, p:862-866,1999.

[7] A.Hamosfakidis, Y.Paker, J. Cosmas "*A study of Concurrency in MPEG-4 Video Encoder*", ICMCS' 98, Austin, Texas, USA. June 1998

[8] S. M. Akramullah, I. Ahmad and M. L. Liou, "*Performance of a Software-Based MPEG-2 Video Encoder on Parallel and Distributed Systems*", IEEE Transactions on Circuits and Systems for Video Technology, Vol.7, No.4, August 1997, pp. 687-695.

[9] D.Farin, N.Mache, Peter H.N. "*SAMPEG, a Scene Adaptive Parallel MPEG-2 Software Encoder*", SPIE Visual Communications and Image Processing, pp. 272-283, 2001.

[10] T. Olivares, F. J. Quiles, P. Cuenca, L. Orozco-Barbosa, I. Ahmad, "*Study of data distribution techniques for the implementation of an MPEG-2 video encoder*", Parallell and Distributed Computing Systems'99. pp.537-542. MIT, Massachusetts (USA). November 3-6, 1999.

[11] Microsoft MPEG-4 Visual Reference Software. July 3rd 2000 Version 2 FDAM1-2.3-001213. Version 2.