PERIODICA POLYTECHNICA SER. EL. ENG. VOL. 40, NO. 1, PP. 53-66 (1996)

DISTRIBUTED PARALLEL COMPUTATION IN VIDEO CODING USING WORKSTATIONS

Sándor BOZÓKI

Department of Telecommunications
Technical University of Budapest
H-1521 Budapest, Hungary
Phone: 36 1 463 2080 Fax: 36 1 463 3266
e-mail:bozoki@com.hit.bme.hu

Received: January 5, 1996

Abstract

In this paper a new approach for coding moving pictures is presented. Because of the large number of calculations, the conventional solution uses tightly coupled multiprocessors working in parallel to achieve real-time processing (encoding 25 - 30 pictures per second). A new idea is to distribute the workload among workstations connected to a network where a software package (e.g. PVM — Parallel Virtual Machine) supports the communication between the machines. In contrast with the present hard wired structures, this loosely coupled system provides more flexibility in coding algorithms and has better cost/performance. The paper describes the main parallel structures already used in video processing, and discusses the possibility of mapping them to this new paralell system. Also, simulations were carried out to examine the performance of the most computationally intensive operations (DCT — Discrete Cosine Transform and motion estimation). The tests were performed on a cluster of SUN Sparc 2s connected via Ethernet. It was experienced that DCT did not show any speed-up because of the extremely low CC ratio. However, motion estimation worked well if either a full or hierarchical search was used. This research work was carried out in 1994 at the Information Theory Group of the Department of Electrical Engineering, Technical University of Delft.

Keywords: video coding, message passing parallel system, PVM, DCT, motion estimation.

Introduction

The transmission of digital video without compression would require a high capacity channel even in low bit-rate applications. As an example, considering a sequence in Common Intermediate Format (CIF), where the resolution for the luminance signal is 360×288 and that for the chrominance components is 180×144, the bit-rate is about 37 Mbit/s if the picture rate is 30 Hz. This image format is used in video phone (H.261 standard [7]) where the upper limit for the bit-rate is 2 Mbit/s.

1. General Video Encoding Algorithm

Since a natural image sequence shows great redundancy in both spatial and temporal directions (the neighbouring pixels[1] on the picture and the consecutive frames[2] are well-correlated), this can be exploited at coding. Most of the video coding standards (H.261, MPEG-1 [8], MPEG-2 [9]) are based on the same DPCM loop indicated in *Fig. 1.* With regard to coding, there are intra and predictive coded pictures.

In the case of the former ones the *Discrete Cosine Transform (DCT)* is used to remove the spatial redundancy, while at the latter also *motion compensation* is done to decrease the temporal correlation.

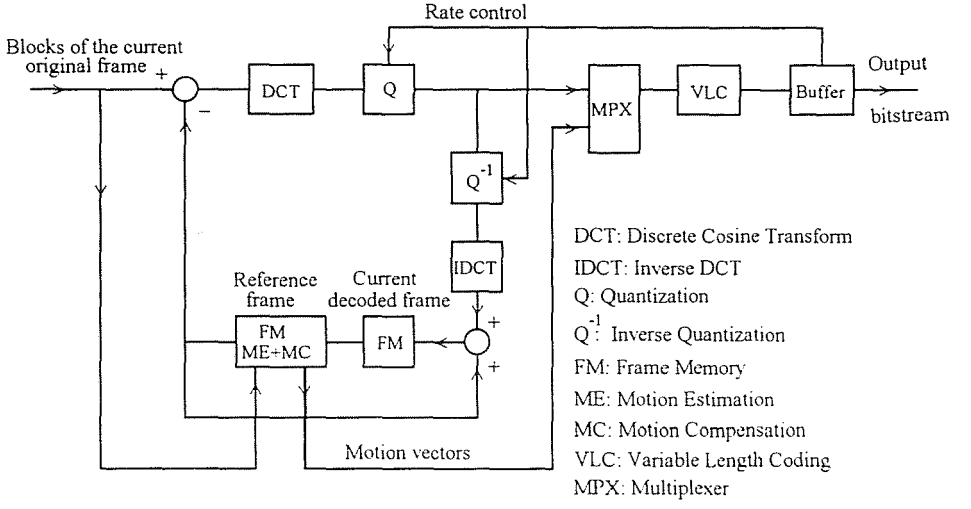

Fig. 1. The DPCM coding loop

The images are segmented into non-overlapping blocks (8×8 or 16×16 pixels in practice), containing the luminance (Y) and chrominance components (Cr, Cb) separately, and the mentioned procedures are applied to them. The DCT transforms the independent blocks into the frequency domain where the coefficients can be represented with fewer bits after quantization than the original pixels in the spatial domain.

[1] A pixel is a picture element or image sample

[2] 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.

The motion compensation is preceded by *block based motion estimation* as follows: Each Y block from the current image is shifted over a search window on a *reference* Y image to find the best match defined by a cost function. The size of the window determines the maximum displacement in each direction. The relative movement of a certain block is described by a motion vector (*Fig. 2*). If all the possible matches are evaluated, the operation is called *full search*, and the best match is always found. This method is very exhaustive, a much more efficient solution is to start the procedure with lower resolution images and finish it with the highest ones. As an example, for a two-step *hierarchical search* the current and the reference picture is decimated in horizontal and in vertical directions by a factor of 2, and a full search is executed. This step results in coarse motion vectors, which will be the initial vectors at the next search. In the second step, images with full resolution are used, but the search area is much smaller for a given block. The method provides a reliable vector field, and the number of calculations is reduced by a factor of 7 per block approximately.

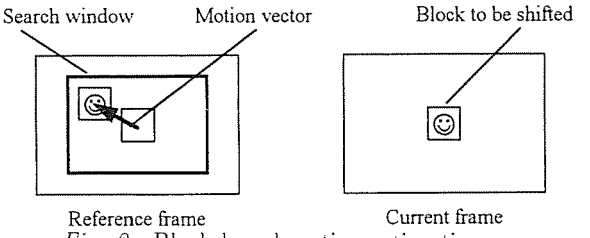

Fig. 2. Block based motion estimation

The reference frame can be the previous or even the next one as well [8,9]. If the best match for a block is good enough, then the corresponding areas are subtracted from each other, that is to say, the actual block is motion compensated (predictive block). In the lack of a good match the block is coded as intra. While motion estimation can be performed either on the original or on the reconstructed reference frame, motion compensation should always be done on the reconstructed image in order to avoid the inequality between the encoder and decoder.

The compression is improved further by assigning code words of variable length (VLC) to special symbols created from the DCT coefficients. The number of bits produced for a frame is dependent on the image contents as well. Consequently, the bit-rate is variable at the output of the VLC. However, in most cases a fixed rate channel is used, thus, a buffer is needed as an interface. Similar buffer can be found also in the decoder. They should run neither into underflow nor overflow during the process, that is why a precise rate control mechanism is required. Normally the

buffer contents has to be examined after processing a row of blocks. The quantization step should be set according to this measurement.

2. Existing Parallel Solutions

2.1 Properties of the Coding Steps

The presented encoder should be flexible in two aspects: On the one hand, it has to satisfy video applications requiring material of different quality, so the coding parameters (e.g. for quantization) should be changeable. On the other hand, the possibility of modifying the algorithms and the structure is also desirable since the standards define the syntax for data and the decoding steps, however, they do not specify the encoder. In respect to the tasks, the following classification can be made:

Low level tasks: This involves DCT and its inverse (IDCT), motion estimation, motion compensation and VLC coding. All of them can be executed independently of the image contents in a regular manner. They are modified seldom.

Medium level tasks: Quantization, coding type decision (intra/ predictive) and rate control belong to this group. They are data dependent, the possibility of adapting them to the signal is needed.

High level task: This involves the control mechanism of the overall system.

In particular the DCT (IDCT) and the motion estimation are computationally intensive procedures although several fast algorithms exist for them. The operations are simple, but they have to be performed many times. For instance, the fast DCT algorithm proposed by LEE [10] requires more than 500,000 multiplications and 1,100,000 additions if it is applied to a CIF image. Considering the same image size, the number of operations is about 200 millions at motion estimation if the full search is used with a maximum displacement of 16 pixels per block. Thus, implementing a video encoder for real-time processing is very difficult. Even powerful workstations are not so fast for this purpose alone, apart from the supercomputer, in every case special hardware is required. Often we do not need such a high-speed processing, we are pleased with a 'reasonable speed', which means, that we do not have to wait for the result for hours. This is very important at the computer simulation phase when we try new algorithms and want to see the results within an acceptable period of time. Furthermore, off-line coding can be satisfactory for many applications where the decoders play pre-recorded video streams (e.g. CD-Interactive with MPEG-1 extension).

2.2 Hard Wired Multiprocessor Structures

Because of the mentioned facts, it is advisable to do the video encoding in parallel. In practice only hard wired multiprocessor structures have been built so far, in which data is routed in a synchronized way. There are three main types of them: *highly-pipeline*, *non-pipeline* and *mixed* structures.

In a highly-pipeline structure only the *operations* are distributed among a number of *special purpose* processors. For instance, one processor does the motion estimation, the other does the DCT, etc. on the whole image. The output of one processor is the input to another one. Although the architecture is optimized with regard to hardware efficiency, it is inflexible, the program in processors cannot be modified in most cases. In a non-pipeline or mixed structure *data* or *data and task* distribution is done among the computing resources.

The spatial distribution of image data is possible since the coding steps are done block by block, and the blocks may be processed independently by keeping certain rules (see slice definition in MPEG). The picture is divided up to equally sized horizontal or vertical stripes of a width of block. All the low level tasks can be performed by *special VLSI chips* (DCT and block matching processors are already available on the market), whereas medium level tasks can be executed by *general purpose* DSPs. The program in DSPs may be modified, what makes this approach somewhat flexible. However, this flexibility is far from the one offered by the pure software solution where also the portability can be provided.

Fig. 3 shows an example of the non-pipeline structure. Similar instances can be found in [4, 5, 6]. Every sub-coder gets a stripe in turn and does all the necessary computation. At motion estimation the reconstructed picture is used as reference which results in smaller error terms at motion compensation. The structure of a sub-coder is similar to the one indicated in *Fig. 1*. The output buffer is located in the multiplexer, its contents can be examined after processing the last N stripes [5]. At motion estimation and compensation interconnection is needed between the neighbouring sub-coders in order to access the correct search windows. If there are more processors in the sub-coders sharing the coding steps, then they have an MIMD (Multiple Instructions Multiple Data) architecture [6], otherwise only an SIMD (Single Instruction Single Data) one [4].

A mixed structure is indicated in *Fig. 4*, the same layout can be found in [3]. The motion estimation and the other low level tasks are done separately by two computing units containing a number of Processing Modules (PM). The original image is used as reference at motion estimation, thus, more accurate vectors are found as opposed to the former case.

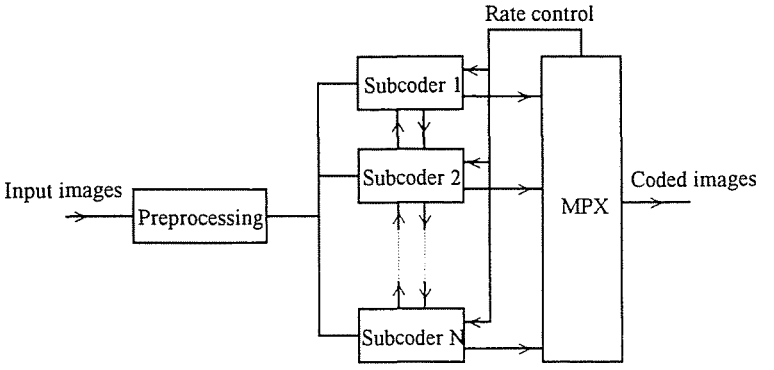

Fig. 3. The non-pipeline structure for video coding

The stripes with the associated search areas are distributed among the PMs in this unit, in which the PMs work independently. However, interconnection is still required between them in the DPCM unit. The main advantage of this structure is the possibility of optimizing all the units separately, although a disadvantage is the need for communication between them. Both structures have a pre-processing stage serving for data preparation, like format conversion or ordering data into blocks.

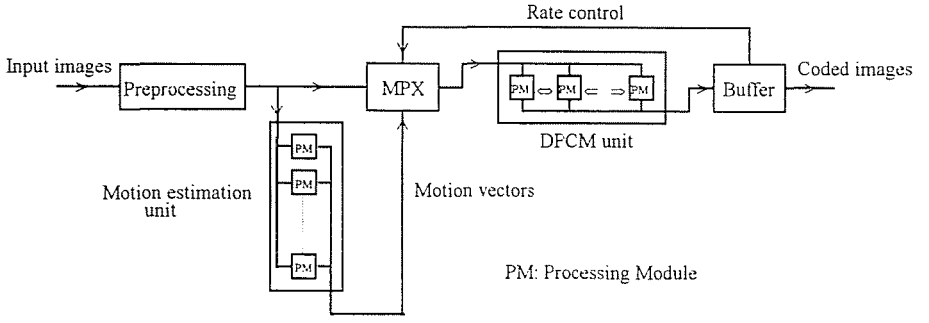

Fig. 4. The mixed structure for video coding

3. The New Parallel Approach

3.1 Workstations as a Loosely Coupled System

A new approach is to distribute image data among workstations connected to a network. This loosely coupled system has a great computational power, the communication between the machines is done by message passing. Because this is the only way for data exchange and synchronization of the operations, the system is called *distributed.* We assume that a multi-user/multi-tasking operating system is installed on the machines.

Unfortunately, there are several factors influencing the performance of the system. The workstations can have different computational power contrasted with the parallel resources in the discussed tightly coupled systems. Also, the load of the machines may fluctuate in time since other processes can join or leave the system, affecting the performance in this way. The communication bandwidth is much lower compared to the case of hard wired solutions. The overhead derived from the scheduling is not negligible either. As a consequence, several problems are not suitable for this kind of parallel implementation, but they work well on a tightly coupled system. In every case the so-called CC ratio (Computation time per byte/Communication time per byte) has to be examined. If this is low (in practice less than 50 [1]), then it is very difficult to achieve a reasonable speed-up. The speed-up has several definitions, in this paper the simplest one will be used which is defined as follows:

$$\text{speed-up} = \frac{\text{time for sequential execution}}{\text{time for parallel execution}}$$

3.2 The Parallel Virtual Machine

The parallel environment can be built by means of the PVM (Parallel Virtual Machine) package, which facilitates the communication and task management between the resources. Under this software all the heterogeneous machines can be viewed as if they were one single machine with a great amount of power. The package is entirely portable since all the main architectures are supported. Besides the mentioned drawbacks, there are several advantages of using such a parallel system:

1. Having a fast network with powerful machines, the system can outperform the supercomputer.
2. Fault tolerance is much higher than that in the hard wired systems. Also, it is easier to recognize if something works improperly.

3. Cost/performance is much better. In addition PVM is a free software.
4. The system is very flexible since only the source code has to be rewritten if new algorithms must be implemented.

The structure of the parallel algorithm has a great impact on the performance of the system. There are some *efficiency factors* which have to be taken into account while developing the code. They are in contrast with each other, an efficient algorithm should find the compromise between them. Let us consider them in turn [1]:

1. The communication between the machines should be minimized because too heavy interconnection may result in a bad speed-up or even no speed-up at all. Because of the behaviour of the real network, increasing the number of machines will not take the speed-up above a certain limit, this will go in saturation because the communication also gets higher.
2. An efficient *load balancing* is needed, which means that all the machines should get an amount of data according to their load and computational power. In our problem the size of data means the number of blocks in a segment. If it is too large, then the requirement above is not met, since some machines can process the same amount of data faster than the others. The resulting computational time is usually greater than it could be obtained because the load balancing is bad. If the size of data is too small, then the communication can be very high although the load balancing is good.
3. The organization of tasks should be done carefully to avoid *data contention*. Data contention is present if several processes require data at a time, but only one can be served at a given moment, the others have to wait. The idle time at any processor node is undesirable because it degrades the overall performance. Too many communications can be the reason for data contention.

3.3 Choosing the Structure of the Encoder

Considering a spatial data distribution, we have two choices in selecting the structure of the distributed encoder. The mixed structure assumes that the motion estimation and DPCM units work synchronously. Since the motion estimation requires significantly more computation, the machines provided with this work should be much faster than those ones doing the DPCM part. Moreover, a fast communication link is required between the two sub-clusters.

If we do not assume anything about the configuration to be used, then the non-pipeline structure is a better choice. Furthermore, it will be shown that doing the DPCM part in parallel does not result in any speed-up. The speed-up will be derived from parallelizing the motion estimation.

The master-slave model was chosen to build up the desired structure (*Fig. 5*). A given slave corresponds to a sub-coder as indicated in *Fig. 3*. It is irrelevant, how many processors the machines contain, because no further parallelization will be done inside them. Thus, the system has a SIMD-like architecture. Data are distributed among the slaves, their task is to do the computation. The communication is done through the TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) sockets. The master does no effective calculations, its duty is to control the system and prepare data to be sent. The latter involves the acquisition of images from a computer file and the necessary segmentation. Also, the master starts the slave executables on different hosts. The user can contact only the master process, all the slaves are hidden.

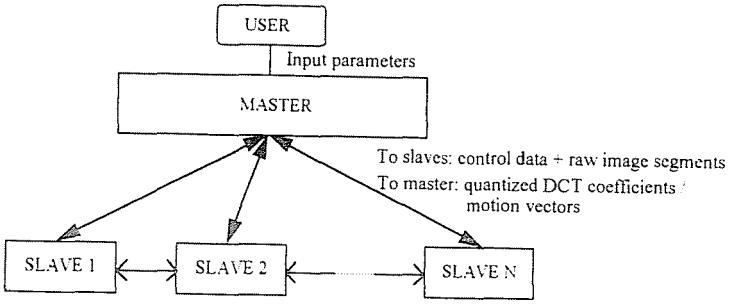

Fig. 5. The master–slave model

In this project only the most time consuming operations (DCT and motion estimation) were examined on this parallel system. At DCT also quantization was carried out as a part of the DPCM loop. The mentioned Lee algorithm was implemented for DCT. At motion estimation the full and the two-step hierarchical search algorithm were tested. We chose the block size of 8×8 at DCT and of 16×16 at motion estimation as they are defined in the standards. *Fig. 6* shows the flowchart of the master and the slave program. They have the following well-distinguishable phases:

Master:

1. **Initialization:** This involves starting the slaves, sending the necessary information to them. such as: how much memory is needed for

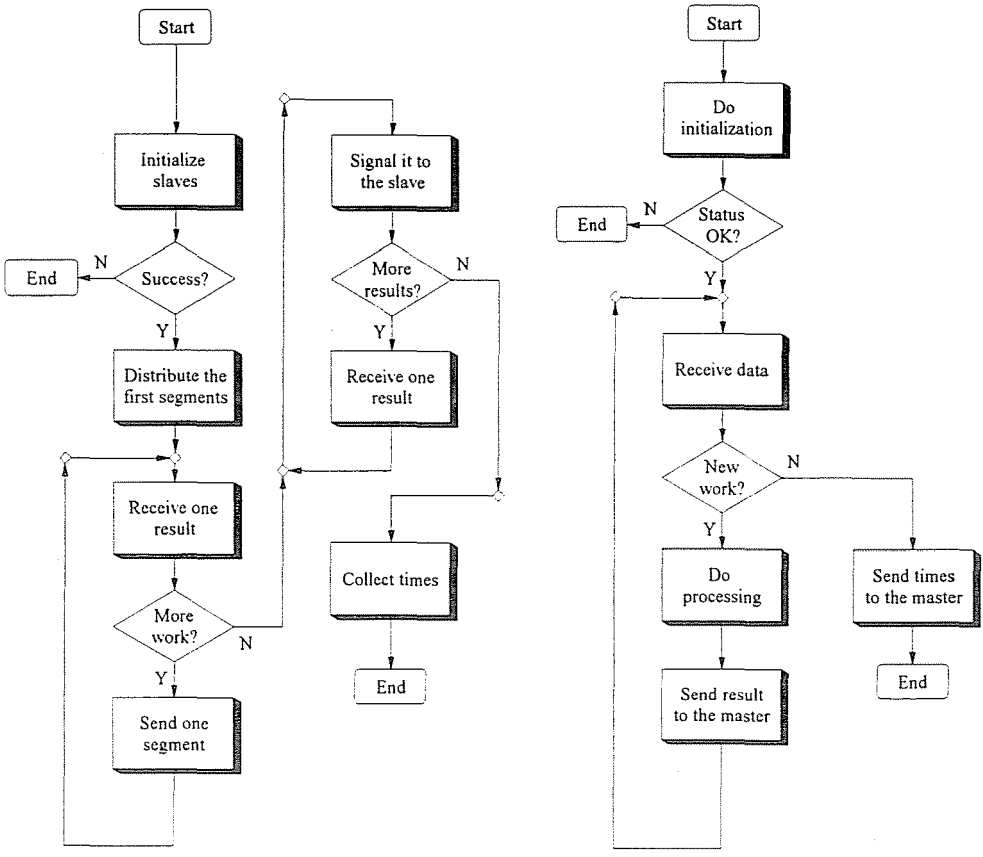

Fig. 6. Flowchart of the master (left) and the slave (right) program

the processing. At motion estimation the maximum displacement and the search type is also transmitted. All the slaves send their status back which indicates whether the initialization was successful or not.

2. **First distribution of segments:** The master assigns a segment to all the slaves in turn and places itself in an idle state. At DCT a segment can be of any size in blocks, but at motion estimation always a row of blocks is transmitted with the associated search area. As a consequence, the slaves do not communicate with each other in this special case.
3. **Sending receiving cycle:** After getting a result, the master sends a new segment to the free slave process if there is some work left, otherwise it signals the end of the processing.

4. **Collecting times:** When all the work has been finished, the master collects the time statistics from the slaves in order to evaluate the performance of the system. This information includes the slaves' computation, idle and communication time.

Slave:

1. **Initialization**
2. **Receiving–computing–sending cycle:** At this stage the slave does all the necessary computation. At the hierarchical motion estimation the decimation takes place on the segments in order to avoid too much data transmission. Also, the filtering would waste time at the master side since normally it should be done on the whole images. It was experienced that this simplification did not degrade the vector field seriously.
3. **Sending the time statistics to the master**

4. Tests and Results

All the tests were performed on SUN Sparc 2 workstations connected via Ethernet. The input images were in Source Input Format (SIF), which means a size of 352×288 for the Y component. This format is derived from CIF by covering the left-most and right-most 4 columns of pixels. By this means, the horizontal image size is a multiple of 16, that is useful at motion estimation.

Fast DCT and motion estimation showed very different parallel performance. In the case of DCT the speed-up was less than 1, irrespective of the number of machines (2 ... 8). The number of segments per frame (4 ... 40) did not affect the speed-up either. The reason for having this result is the low CC ratio, which was measured as 1 ... 4. It was calculated simply as the ratio of the slaves' average computation and communication time. *Fig. 7* illustrates the time statistics of the slaves in the case of transforming one frame. The communication overhead is almost constant, however, the average period of the idle state increases with the number of machines. This is because of the master's inability to serve the slaves in time, which is the source of data contention.

The CC ratio might be larger by using a faster network (e.g. ATM). As it is known, the conventional Ethernet has a maximum throughput capacity of 10 Mbits/s only. However, in our opinion the speed-up will not be satisfactory in that case either since the communication start-ups take some time, which can be close to the computation time.

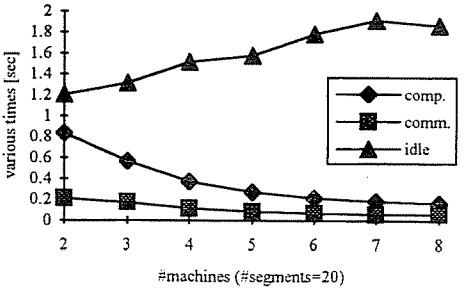

Fig. 7. Average computation, communication and idle time of the slaves

However, the performance of the motion estimation is satisfactory in both cases (full and hierarchical search). Table 1 summarizes the times needed per frame on one SUN Sparc 2. The maximum displacements are typical values used in video coding.

Table 1
Times needed per frame for motion estimation on one SUN Sparc 2.

Max. displacement	Full search [sec]	Hierarchical search [sec]
8	30	5
16	115	12

Fig. 8a shows the speed-up for the full search. The CC ratio is about 80 – 90, thus, an almost linear speed-up is achievable even if the maximum displacement is small (e.g. 8). An even better result can be obtained if the search range is increased. The speed-up for hierarchical search is still satisfactory especially if the maximum displacement is 16 or larger (*Fig. 8b*). These results make possible to encode video in a spatially parallel way. A given slave has to process an image segment entirely, including all the coding steps. Unfortunately, the communication overhead introduced between the slaves may degrade the performance. Further research will be carried out to examine its influence on the overall speed-up.

5. Conclusion

In this paper the possibility of encoding digital video on a distributed parallel system was examined. The advantages and also the drawbacks of this new method were discussed. We considered a spatial distribution of image data. As it was shown, the computing resources should process an

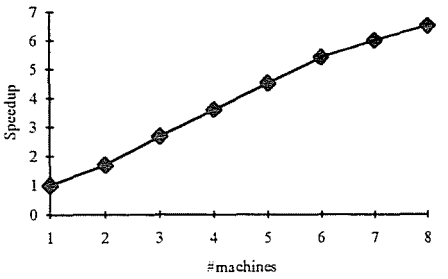

Fig. 8a. Speed-up for full search (max. displacement is 8)

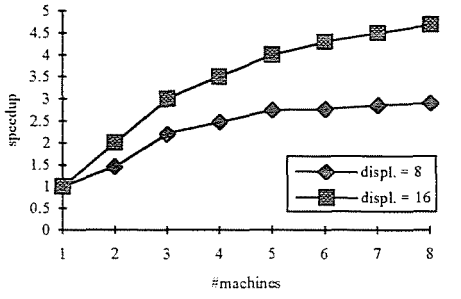

Fig. 8b. Speed-up for hierarchical search at different displacements

image segment entirely to produce a resonable speed-up. The separation of the coding tasks is useless, because the DPCM part has an extremely low CC ratio, resulting in no speed-up at all. The importance of this topic implies the possibility of the off-line encoding in an efficient way. Furthermore, simulations carried out in the research life can be done faster, facilitating the quicker evaluation of new methods by this means.

Acknowledgement

The author would like to thank Dr. R. L. Lagendijk and S. Westen for their kind help, and Tempus for financing this project.

References

1. NING TANG: Parallel DCT Simulation with Workstations, Taakverslag, Technische Universiteit Delft, Faculteit der Elektrotechniek, Vakgroep Informatietheorie.
2. PVM 3.0 User's Guide and Reference Manual.
3. BOUVILLE, C. et al.(1993): A Real Time Testbed for MPEG-1 Video Compression, *SPIE*, Vol. 2094 pp. 205–212.
4. DE SA, L. et al.: Parallel Architecture for Real-time Video Communication, *SPIE*, Vol. 1360 Visual Communications and Image Processing '90, pp. 380–387
5. COTTON, A. D. R. – WELLS, N. D.: Coordination of Parallel Processing within HDTV Bit-rate Reduction Codecs, *International Workshop on HDTV '92*, Tokyo, pp. 63.1–63.8.
6. WILBERG, J. et al: Hierarchical Multiprocessor System for Video Signal Processing, *SPIE*, Vol. 1818 Visual Communications and Image Processing '92, pp. 1076–1087.
7. Video Codec for Audiovisual Service at px64 kbits/s ITU-T Rec. H.261 (1990).
8. Coding of Moving Pictures and Associated Audio, ISO 11172 (1992).
9. Generic Coding of Moving Pictures and Associated Audio, ISO/IEC 13818 (1994)
10. LEE, B. G.: A New Algorithm to Compute the Discrete Cosine Transform, *IEEE Transaction on Acoustic and Speech Signal Processing*, Vol. ASSP-32, pp. 1243–1245, Dec. 1984.