ELSEVIER

Signal Processing: *Image Communication* 16 (2001) 431–444

SIGNAL PROCESSING: IMAGE COMMUNICATION

www.elsevier.nl/locate/image

A distributed adaptive block matching algorithm: Dis-ABMA

F. Vermaut*, Y. Deville, X. Marichal, B. Macq

Dept. d'Ingénierie Informatique/Dept. d'Electricité, Place Sainte Barbe 2, 1348 Louvain-la-Neuve, Belgium

Received 18 December 1998

Abstract

In the context of motion estimation for video sequences processing, variable block size algorithms, like the Adaptive Block Matching Algorithm (ABMA), have been proposed to match better "objects in motion" compared to the classical BMA. However, the variable block size grid derivation and the related motion estimation relies on a regularization process which implies heavy iterative and inter-dependent computations. Though the parallelization of the BMA is straightforward, the ABMA needs a deeper analysis before its implementation in a distributed environment: this is the goal of this paper. We first designed a modelization of the motion estimation of ABMA. This model can lead to several different distributed versions. A specific distributed model, with one master and several slaves, is then described. An implementation of this model has been realized and experimentations demonstrate a linear speedup with respect to the number of processors.

Keywords: Video processing; Video coding; Motion estimation; Distributed algorithms; PVM

1. Introduction

Motion estimation (ME) algorithms are useful tools in many applications related to video signals. Such applications include video compression, video indexing, object tracking, etc. In a video coding framework, ME aims at detecting motion from one image to another. Such a motion information, possibly represented as vectors, can be used for temporal prediction of successive images [15] and for compression. The MPEG [3,8] coders family is based on this principle.

The Block Matching Algorithm (BMA) [5] is a very popular ME algorithm because of its good balance between result accuracy and algorithm complexity. BMA decomposes images in blocks of fixed size. For each block of the current image, one searches a similar block in the previous image. The MPEG-1, MPEG-2 [8] and MPEG-4 [3] coders use the Block Matching Algorithm for their motion estimation step.

The Adaptive Block Matching Algorithm (ABMA) [9,12] is an improvement on BMA. In BMA, the size of the blocks is crucial: large block sizes give rise to what are called "blocking artifacts"; small block sizes increase the noise sensitivity and give rise to non-zero vectors for null motions. The idea of ABMA is to locally adapt the size of the blocks only where it is necessary to match better objects. The decomposition is not anymore purely block based but more "*object in motion*" oriented: large blocks are used in areas with uniform motion, while small blocks are

* Corresponding author. Tel.: + 32-10-473150; fax: + 32-10-450345.

E-mail addresses: fv,yde@info.ucl.ac.be (F. Vermaut, Y. Deville), Marichal,Macq@tele.ucl.ac.be (X. Marichal, B. Macq).

0923-5965/01/$ - see front matter
PII: S0923-5965(00)00008-4

applied along object borders where contradictory movements arise. At an equivalent bit-rate, ABMA overcomes BMA performances in terms of quality after decoding. The main disadvantage of ABMA is a slow estimation step which prevents real-time applications.

Objectives and results. The main objective of this paper is to propose a distributed version of ABMA that can speed up the coding time. The obtained results are the following:

- An abstract specification of ABMA is provided.
- A distributed modelization of ABMA is designed. This model can lead to different distributed versions.
- A distributed version, based on the PVM platform [7], is implemented in terms of one master and several slaves.
- Experimentations demonstrate a linear speedup with respect to the number of processors.

Related works. Parallelization and/or distributed versions of motion estimation algorithms have already been explored, but mostly through a massive parallelization of BMA, by means of SIMD architecture (see for example [2,11]). Our approach does not make any hypothesis on the underlying architecture. Although our particular implementation uses a network of conventional workstations, our model also covers such massive parallel approaches. This paper is an expanded version of [16].

Outline of the paper. Section 2 presents the principles of the Adaptive Block Matching Algorithm and defines the underlying mathematical notions. Section 3 describes the Distributed ABMA and Section 4 exposes experimental results. Section 5 concludes the paper.

2. Background: the Adaptive Block Matching Algorithm

As briefly stated in Section 1, block-based ME techniques consider a pair of images extracted from a sequence. From these two images the algorithm tries to find the motion undergone by regions of the images. Those movements are then expressed in terms of vectors.

The Block Matching Algorithm is based on block decomposition of one image and on searching a corresponding block in the previous image of the sequence. Since the Adaptive Block Matching Algorithm is an improvement on the Block Matching Algorithm, the section starts with an overview of BMA.

2.1. *Block Matching Algorithm* (*BMA*)

The BMA [5] uses a decomposition of an image into blocks of fixed size. For each block of the image, the previous image is searched for a similar block. A similar block means a block that gives rise to a minimal error. The error between similar blocks is measured by means of the mean absolute error (MAE).

Consider two images P1 and P2. These are matrices of pixels. Consider a block B of size $k \times k$, submatrix of P2. The BMA looks for a displacement of this block from the image P1 to P2. This displacement will be the difference between the positions of B and a similar block inside P1. The search is usually restricted into a window of P1 centered around the position of B in P2.

For a given position of a block in the search window in P1 the MAE is defined as follows [13]:

$$\mathrm{MAE} = \frac{\sum_{i,j=1}^{k} |B(i,j) - B'(i,j)|}{k^2}, \tag{1}$$

with k the size of the blocks, B' the block in P1, and B the block in P2.

The objective of the BMA is to find, inside the search window, the position of a block which minimizes the MAE. Inside the search window (Fig. 1) the number of positions for a block of size k is equal to $(2\mathrm{d}h + 1)(2\mathrm{d}v + 1)$, with $\mathrm{d}h$ the distance for horizontal search and $\mathrm{d}v$ the distance for vertical search. Since the complexity of one computation of MAE is $\mathrm{O}(k^2)$, the temporal complexity for BMA on one block is then $\mathrm{O}(k^2\,\mathrm{d}h\,\mathrm{d}v)$.

The value of the minimal MAE does not always provide sufficient information on the "certitude" of the movement of the block.

Although not present in the BMA, the concept of motion certitude is introduced to measure this idea

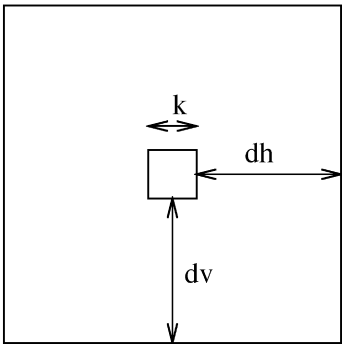

Fig. 1. Search window around a block.

of certitude. A mathematical definition for the motion certitude is given by

$$\mathrm{MC} = \frac{\sum_{i=1}^{N_s} |\mathrm{MAE}_i - \mathrm{MAE}_{\min}|}{N_s} = \frac{\sum_{i=1}^{N_s} \mathrm{MAE}_i}{N_s} - \mathrm{MAE}_{\min}, \quad (2)$$

with N_s the number of positions for the block in the search window $(2dh+1)(2dv+1)$ and $\mathrm{MAE}_{\min}$ the minimal MAE among the MAEs for all the positions. The second equality is correct because MAEs are always positive numbers. The motion certitude thus measures the mean of the distance between the best MAE and the others.

The mathematical definition of the motion certitude is illustrated by the example of Fig. 2. For the block on the periphery (Fig. 2(a)), the mean of the difference with the minimal error is high because all errors are high except the minimal one. This minimal error is very low due to the similarity with the considered block. But for the internal white block (Fig. 2(b)), many errors are similar and very low and the mean of the differences is thus also low. The "motion certitude" represents in a good way the trust one can have in the motion estimated by BMA.

The specification and the algorithm of BMA are given in Fig. 3.

2.2. Adaptive Block Matching Algorithm (*ABMA*)

The ABMA is an improvement on the BMA. In BMA, the initial size of the blocks is crucial. Big blocks give rise to what are called "blocking artifacts". Small blocks increase the sensitivity to noise and give rise to non-null vectors for null movements. So the idea of ABMA is to decrease the size of the blocks only where it is necessary to look for a more precise movement. The decomposition is not purely block-based anymore, but more "*object in motion*" oriented.

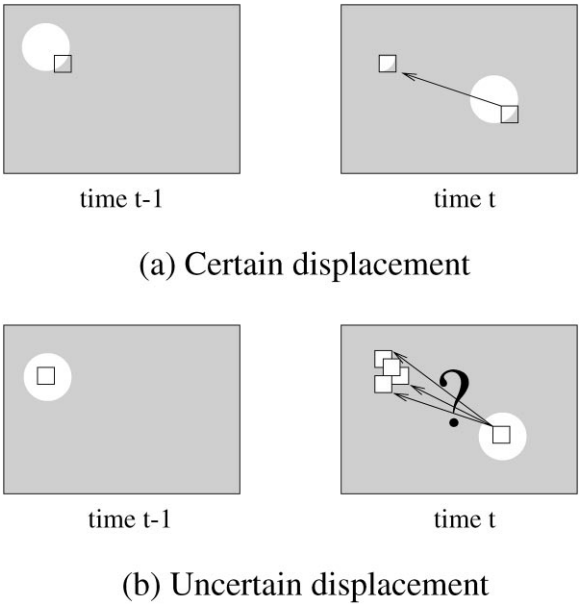

Fig. 2. Motion certitude.

ABMA is composed of 3 steps, each working on a pair of images:

1. A global motion estimation.
2. The determination of a mask of changes.
3. A local motion estimation, relying on BMA.

The complexity of the two first steps is much lower than the complexity of the third step. Hence only this last step will be distributed. For the sake of completeness, the two first steps are quickly presented. For more information, please refer to [9,12,13].

2.2.1. Global motion estimation

The goal of this step is to estimate the movement of the camera: panning and zoom. After such an estimation, the first image is compensated with the global vector and the global zoom factor. The omnipresent component of the camera motion is

```
Algorithm BMA
pre :    P1 an image, B a block in image P2
         dh, dv : the size of the search window in P1
post :   vecx, vecy : motion vector of a block in the search window of P1,
                      minimizing the MAE with B
         bestMAE : associated minimal MAE
         mc : associated motion certitude

  begin
1    mc:=0;
2    bestMAE:=+∞;
3    vecx,vecy:=0;

     {Comparison of B with all positions in search window. Position of B = (Bx, By)}
4    for each i in −dh..dh do
5       for each j in −dv..dv do
6       begin
           {Compute MAE between B and block of P1 at (Bx + i, By + j)}
7          MAE := COMPUTE_MAE(B, P1(Bx + i, By + j));
8          mc:=mc+MAE;
           {Keep minimal MAE}
9          if (MAE < bestMAE) then
10         begin
11            bestMAE := MAE;
12            vecx := i;
13            vecy := j;
14         end
15      end
16   mc := mc/(dh*2+1)*(dv*2+1);
17   mc := mc−bestMAE;
  end
```

Fig. 3. Specifications and algorithm for BMA.

thereafter suppressed and gives the possibility to find locally null motions. It has been proven in the literature that such a two-stage global/local motion estimation drastically improves the quality of motion estimation [4,14].

The estimation of both translation and zoom is made conjointly in a unique process.

2.2.2. Mask detection

To reduce the amount of computations of the next step, a cheap method is used to detect where the motion can be considered as null. The algorithm computes a mask of changes. This is a binary image which contains for each pixel an information telling if the pixel changed between the globally compensated image P1 and P2. The algorithm compares each pixel from the compensated P1 with its corresponding pixel in P2 and, then, executes some filtering.

By the use of this mask it is easy to detect which blocks of the images did not undergo any motion.

2.2.3. Local motion estimation

Local motion estimation is the main part of ABMA. It gives as a result a multigrid (see Fig. 4), and for each block of the multigrid a motion vector. To obtain the final motion field, the global component must obviously be added to the local vectors.

As this step is computationally expensive, a distributed version will be developed in the next section. This step must therefore be precisely specified. The specifications and the algorithm of the local motion detection are given in Fig. 5.

After having decomposed image P2 in a grid of blocks with the same original size, the first step is *to refine the mask of changes* to detect which blocks can be directly considered as having a null motion.

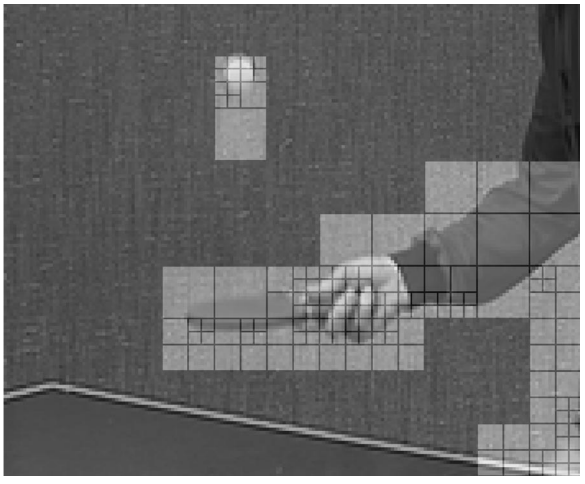

Fig. 4. Resulting multigrid of local motion estimation.

This is easy because a block, for which a large fraction of its pixels are unchanged, can be considered as unchanged. The REFINE MASK operation gives as result the list of blocks for which a motion vector must be computed. These blocks are called "untreated" blocks.

The next step consists of a loop of four substeps. The number of iterations is a parameter specified by two quantities: the maximal size and the minimal size of blocks. The initial size of the blocks is specified by sizemax. If no "good" motion vector can be found for a block, the block is split into smaller blocks. The process is repeated until the blocks are too small (sizemin).

Within the loop, four processes are applied to all the untreated blocks of current size.

- *BMA*. Let us recall that the temporal complexity of BMA is $O(k^2\,dh\,dv)$, where k is the size of block B.
- MC_TREATMENT. When the motion of a block, obtained with BMA, is uncertain (i.e. its motion certitude is too low), it is sometimes preferable to choose the vector of one of its four neighbors (north, east, south, west). This is done only if the vector of the neighbor, after application to the block, does not induce too large an error (MAE). The rationale of the treatment is based on the idea that, between two images, adjacent blocks often undergo similar movement. This step has a temporal complexity of $O(k^2)$.
- FILTERING. The purpose of the non-linear filtering is to smooth the differences between motion vectors of adjacent blocks. This operation is mainly relevant in a video coding framework where one may wish to lower the entropy of the motion vector field. When certain configurations of vectors appear, the motion vector of one of the 8 classical neighbors is preferred if the vector, after application to the block, does not again induce too large an error. This step has a

```
Algorithm LOCAL MOTION ESTIMATION
pre :    P1, P2 : image
post :   A multigrid (with a motion vector for each of its blocks)
         that represents the movement of blocks from P2 to P1

  begin
1     sizeblock := sizemax;
2     REFINE_MASK;
3     while (sizeblock ≥ sizemin) do
4     begin
5        for each (untreated blocks) do BMA;
6        for each (untreated blocks) do MC_TREATMENT;
7        for each (untreated blocks) do FILTERING;
9        if (sizeblock > sizemin) then
10          for each (untreated blocks) do SPLIT_OR_KEEP;
11       sizeblock := sizeblock / 2;
12    end
  end
```

Fig. 5. Specifications and algorithm for local motion detection.

temporal complexity of $O(k^2)$. Notice that this temporal complexity might be transformed into storage requirement (i.e. $O(dh * dv)$ per block) since MAE has already been calculated for every possible block position in the BMA step.

- SPLIT_OR_KEEP. The decision to split or not to split a block is taken here. The goal is to look if the vector is acceptable, after motion certitude treatment and non-linear filtering. The criterion to keep the block is that its motion vector induces an acceptable (small) error (MAE). This means that it is representative of the movement of the whole block. Otherwise the block is split into 4 smaller blocks with half the size of the father block. These blocks inherit the vector of the father and the blocks for which this vector is not representative are marked as "untreated". These smaller blocks move then to the next iteration.

At the end of the loop, all remaining untreated blocks keep their best vector.

3. Distributed ABMA

3.1. The model

To realize a distributed version of the ABMA algorithm, a model of the problem must first be settled. From this model, various sequential or distributed architectures can be deduced. A distributed version must obviously give the same results as the sequential one. One may already see that the main "object" in ABMA is the *block*.

In the sequential LOCAL MOTION ESTIMATION it can be observed that:

1. A motion vector is known for all the blocks that are not "untreated".
2. The BMA operation performed on a block is an independent operation.
3. For the treatment of the motion certitude (MC_TREATMENT) of a block, the result of BMA on this block must be known (motion vector, MAE and motion certitude). Moreover, the result of BMA must also be known for its four neighbors.
4. The FILTERING step on a given block uses the result of the motion certitude treatment for the considered block and for its eight neighbors.
5. In the SPLIT_OR_KEEP step of a block, no information is required about the neighbors.
6. In the beginning of every iteration step, all untreated blocks have the same size.

Some synchronization points can be distinguished in this algorithm, separated by transitions (computations) and transition conditions. The ABMA will be modeled by means of a finite state automaton [6,10], in terms of states in the life of blocks, and transitions between these states (see Fig. 6).

In the sequential algorithm, there are 4 points where a block ends a treatment: after BMA, after the motion certitude treatment, after filtering, and after the split decision step.

The initial state of a block results from 'MASK_REFINE". Some blocks are untreated and some are already detected as having a null motion. For the untreated blocks, no information is known. The initial state of these blocks is called "*Unknown*". The other blocks are in a state called "*Done*" because their treatment is over. An untreated block may quit its initial state and fall into the "*Known*" state once BMA has been applied. Then its vector, its MAE and its motion certitude are known. The next transition is the motion certitude treatment of the block. The block falls then into a state called "*Propagated*". As explained earlier, the propagation of vectors is the root of motion certitude treatment, hence the name of this state. Information about the neighbors is needed and a synchronization state is required. Filtering can then be realized. As the SPLIT_OR_KEEP treatment does not require any synchronization with the neighbors of a block, there is no need for a specific state after filtering. The filtering and the split decision can thus be applied on a block in the *Propagated* state, leading the block into the *Done* or *Split* state.

From this model, efficient sequential versions can be implemented, as well as different distributed versions. A particular distributed solution will be described in Section 3.3.

3.2. State transitions

Following the life of a specific block brings information about the state transitions. Assume that the initial state is *Unknown* (i.e. the REFINE_MASK

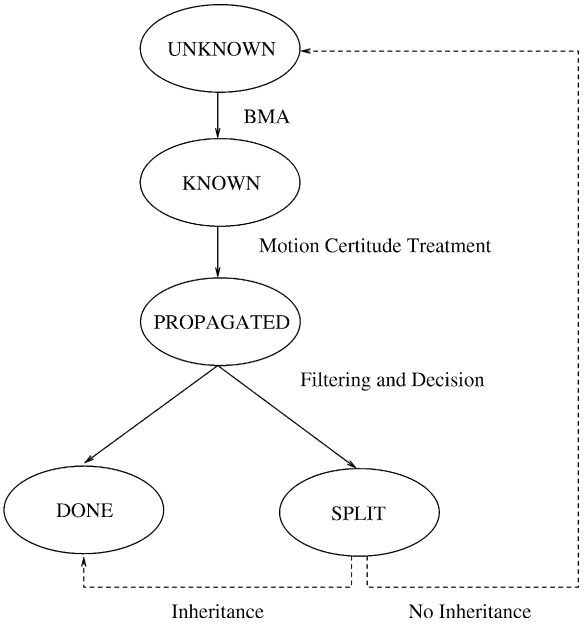

Fig. 6. State diagram.

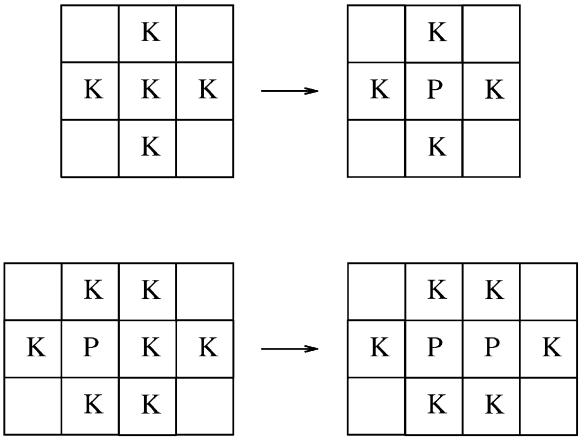

Fig. 7. *Known* to *Propagated* transition conditions.

operation did not detect a null motion). After the application of the BMA process, this block goes into the *Known* state. In order to apply the MC_TREATMENT and become in the *Propagated* state, the BMA process must have already been applied on the four neighbors of the considered block. Hence, the requirement that the four neighbors are in the *Known* or in the *Propagated* state, as illustrated in Fig. 7. Obviously, a block in the *Propagated* state must keep the results of the BMA process.

In order to apply the non-linear filtering on a block in the *Propagated* state, its eight neighbors must be in the *Propagated* state, or in later state (*Done* or *Split*). A block in the *Done* or *Split* state must thus also provide the vector value resulting from its MC_TREATMENT.

The proposed model does not require to treat all the blocks of a given size before treating blocks of smaller size (obtained after splitting). As a consequence, if a block in the *Known* or *Propagated* state has a neighbor of larger size, one should wait either for the larger block to fall into the *Done* state, or for the larger block to become split so that the corresponding (smaller) neighbors fall into the appropriate state. An example is provided in Fig. 8.

One could easily show that it is impossible, for a block, to have a neighbor of larger size in the *Unknown* or *Known* state. Moreover, if the size of the larger block is "two steps" larger, then the large block can only be in the *Done* state (see Fig. 9).

3.3. Master–Slaves instantiation of distributed ABMA

As BMA is the most expensive computation, we chose to distribute this step between slaves. So this step is realized on several blocks in parallel (see Fig. 10). The slave can thus perform one action: BMA. A master [1] is used to distribute the different blocks, to receive the results of computation of the slaves and to perform motion certitude treatment and filtering. Those two last steps are performed as soon as the necessary information is available. A formal description is given in the next sections.

In this model, the master and all the slaves possess the two images: P2 and the globally compensated P1. This is the only information for the slaves.

For the master, four data structures are needed. The master must first possess the entire multigrid, which will contain the final result of ABMA. Then,

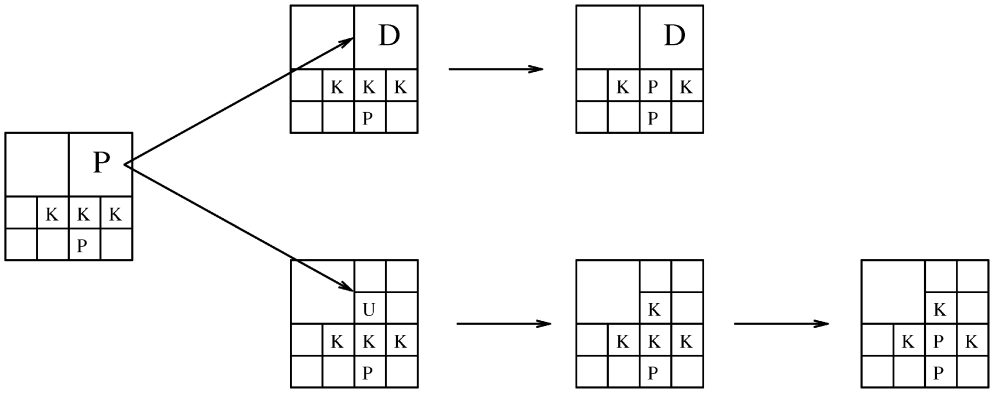

Fig. 8. *Known* to *Propagated* transition conditions.

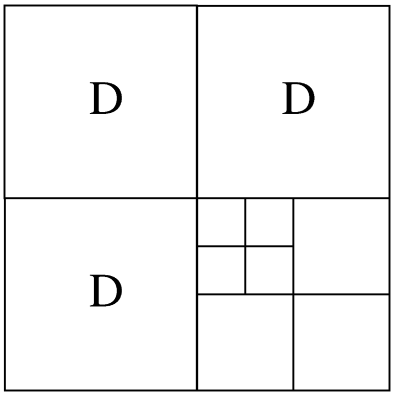

Fig. 9. Two neighbors with more than one step of size.

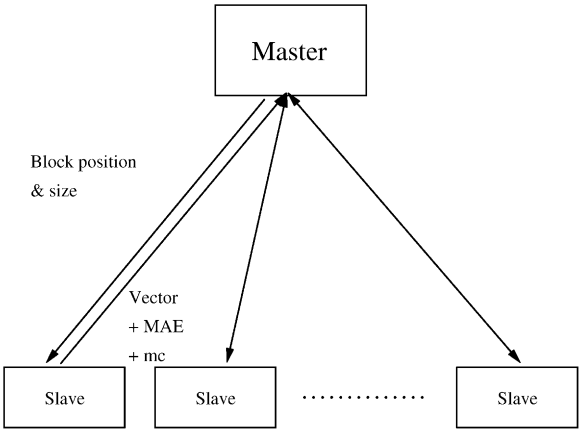

Fig. 10. Master–Slaves structure.

two lists are needed: one for the untreated blocks, thus in the *Unknown* state, and one for the free slaves. The fourth structure is needed to record the blocks currently involved in a BMA computation. The master has to know which block a slave possesses: a list of couples ⟨slave,block⟩ is used and is called "block2slave".

MG = the Multigrid.

WWL = Waiting Works List, list of blocks to be treated by the slaves.

WSL = Waiting Slaves List, list of free slaves.

block2slave = list of pairs ⟨slave, block⟩ indicating which block is handled by which slave.

For each block, the following information is recorded:

b.size = The size of block b

b.State = The current state of bloc b

$b.\mathrm{MAE}_s$ = The MAE of bloc b at state $s \in \{\mathrm{K(NOWN)}, \mathrm{P(ROPAGATED)}, \mathrm{D(ONE)}\}$

$b.V_s$ = The Motion Vector of bloc b at state $s \in \{\mathrm{K(NOWN)}, \mathrm{P(ROPAGATED)}, \mathrm{D(ONE)}\}$

$b.\mathrm{MC}_s$ = The Motion Certitude of bloc b at state $s \in \{\mathrm{K(NOWN)}\}$

Invariant: $\forall b: block \quad b \in WWL \Leftrightarrow b.State = Unknown$

3.4. Formal specifications

This is the central part of the modelization. In order to specify correctly the "block2slave" data structure, the state diagram is enriched with a new state: WAITING. During the BMA operation, a block will be in the WAITING state (see Fig. 11). The formal specifications are given in Fig. 12. The *precondition* specifies the condition a block must satisfy in order to realize the transition. The *postcondition* specifies the results (on the block and on the data structure) of the transition. Finally, the *triggering* information specifies when the transition can be triggered. In other words, this characterizes conditions, on the environment of a block, that must be fulfilled before the transition can be performed.

Some additional properties can easily be deduced. If one of the four neighbors of a block in the *Known* state has a larger size, then the state of this larger block will necessarily be *Propagated* or *Done*. Similarly, if one of the eight neighbors of a block in the *Propagated* state has a larger size, then its state will be *Done* if it is one of the four vertical or horizontal neighbors. Otherwise its state will be either *Propagated* or *Done*. These properties can be used to simplify the trigger conditions in the implementation.

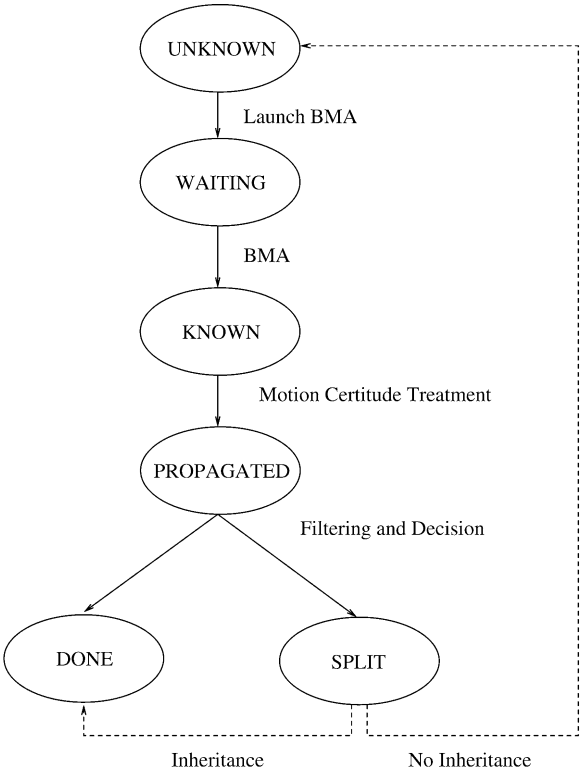

Fig. 11. New WAITING state.

4. Experimental results

The proposed model of Distributed ABMA, particularized in a master–slaves structure, has been implemented in C using a network of conventional workstations (SUN computers). The PVM platform [7] has been used for distribution.

For a given execution, the number of processors and the number of slaves are fixed. Moreover, one processor is assigned to the master, and distinct processors for the different slaves. Hence, for p processors, $p-1$ slaves are used.

For each experiment, we measure the *global execution time*. As the master launches all the slaves at the beginning of its execution and kills them at the end, the global execution time is also the execution time of the master, including its idle times when it waits for slaves results. Such a global execution time thus includes all the computations and communications. For each slave, we record the *slave computation time*. Such a slave computation time does not include the idle time of that slave, waiting for some work to perform. We also define the *slave idleness* as the ratio between (global execution time minus slave computation time) and global execution time. The slave idleness thus measures the percentage of idleness of a slave. The *speedup* is defined as the ratio between the execution time of the sequential version, and the global execution time of the distributed version. The resulting speedup is thus dependent on the number p of processors. We finally introduce the *efficiency*, ratio between the speedup and the number of processors (1 master and $p-1$ slaves).

Our experimentations have the objectives to compare Dis-ABMA with the sequential ABMA by analyzing the influence of the number of slaves on the global execution times. The experimentations

LAUNCH BMA
$\forall b : block, e : slave$
pre : $b.State =$UNKNOWN
post : $b.State =$WAITING $\wedge\ WSL = WSL \setminus \{e\}$
$\wedge\ WWL = WWL \setminus \{b\}$
$\wedge\ block2slave = block2slave \cup \{< e, b >\}$
trigger : $\exists e \in WSL$

BMA COMPLETED
$\forall b : block, e : slave$
pre : $b.State =$WAITING
post : $b.State =$KNOWN $\wedge b.V_K =$ vector value
$\wedge\ b.MAE_K =$ associated MAE
$\wedge\ b.MC_K =$ motion certitude
$\wedge\ WSL = WSL \cup \{e\}$
$\wedge\ block2slave = block2slave \setminus \{< e, b >\}$
trigger : Reception of slave results

MOTION CERTITUDE TREATMENT
$\forall b : block$
pre : $b.State =$KNOWN
post : $b.State =$PROPAGATED
$\wedge\ b.V_P =$ vector value after motion certitude treatment
$\wedge\ b.MAE_P =$ associated MAE
trigger : $\forall n :$ neighbor of 4-connectivity
$[n.size = b.size \wedge (\ n.State =$ KNOWN $\vee\ n.State =$ PROPAGATED $)]$
$\vee\ [n.size > b.size \wedge n.State =$ DONE $]$

FILTERING AND DECISION
$\forall b : block$
pre : $b.State =$PROPAGATED
post : $(b.State =$DONE $\wedge b.V_D =$ final vector value after filtering
$\wedge b.MAE_D =$ associated MAE $)$
$\vee\ (b.State =$ SPLIT $\wedge\ WWL = WWL \cup \{b_1, b_2, b_3, b_4\})$
with b_1, b_2, b_3, b_4 four new smaller blocks.
trigger : $\forall n :$ neighbor of 8-connectivity
$[n.size = b.size \wedge\ (n.State =$ PROPAGATED
$\vee\ n.State =$ DONE $\vee\ n.State =$ SPLIT $)]$
$\vee\ [n.size > b.size \wedge n.State =$ DONE $]$

Fig. 12. Formal specification.

also aim at comparing the benefits of a distributed version on images with different motion complexities and analyzing the effective distribution of work between the slaves and the impact of using a distributed version on a complete sequence of images.

The studied implementation of Dis-ABMA (master–slaves structure) has not only an experimental value, but the experimentations show that such a solution can be used in practice.

4.1. Number of slaves and motion complexity

One processor for the master and up to 19 distinct processors for the different slaves have been used for the experimentations. Figs. 13 and 14 provide the global execution times (in s) for the sequential ABMA and for the distributed version (Dis-ABMA), as well as the speedup. The given measures were obtained on a pair of images extracted from the complex MPEG-4 sequence "Stefan"

(see Fig. 15). In the chosen images, the mask of changes is large (white surfaces), hence requiring subsequent complex computations.

These experiments show a quasi-linear speedup. The efficiency is about 70%. Due to communication overhead, the distributed version with 2 processors (1 master and 1 slave) is slower than the sequential ABMA. When the number of slaves is high (more than 9 in this experiment), the efficiency factor decreases. This is explained by the fact that the number of slaves is too high compared with the work the master can distribute simultaneously, and the amount of work of the master.

The same experiments were performed on a pair of images from a low motion complexity sequence, frames 196GC/199 from the MPEG-4 "Akiyo" sequence (see Figs. 16 and 17). The distributed version performs very well when the number of slaves is not too high (less than 5); the speedup is linear, and the efficiency is about 60%. But for such low motion images, the amount of work to be distributed is not very high and it is useless to use many slaves, which explains the almost constant global execution time of Dis-ABMA with many slaves.

The above two experiments show the advantages of Dis-ABMA, compared to sequential ABMA, for complex sequences as well as for non-complex sequences. The number of slaves to involve in a computation could be dynamically determined as soon as the complexity of the motion is known owing to the mask of changes.

4.2. Equity between slaves

The load balancing technique is implicit in the formal specifications of the transition conditions, particularly for our master–slaves model. By using a *queue* of free slaves and a *queue* of treatable blocks, a priority order is introduced in the choice of the next block to be treated and of the slave that will treat it. A fast slave will thus appear very frequently in the list of free slaves and will treat many blocks.

Fig. 18 presents the different slaves' computational time, the global execution time and the average slave idleness for the "Stefan" images. The results are only shown for 1 to 10 slaves. One can observe an equity between the slaves computation times. The average idle time is very low, except for 10 slaves. As already explained earlier, the amount of computation to distribute begins to be insufficient when the number of slaves become too large (10 in this example).

Sequential ABMA	Nb of Processors	Nb of Slaves	Dis-ABMA (sec.)	Speedup
31.23	2	1	33.91	0.91
	3	2	17.12	1.82
	4	3	11.81	2.64
	5	4	8.85	3.53
	6	5	7.02	4.45
	7	6	5.89	5.30
	8	7	5.11	6.11
	9	8	4.49	6.95
	10	9	4.00	7.81
	11	10	3.82	8.17
	12	11	3.45	9.05
	13	12	3.21	9.73
	14	13	3.10	10.07
	15	14	3.01	10.38
	16	15	2.80	11.15
	17	16	2.85	10.96
	18	17	2.68	11.65
	19	18	2.39	13.07
	20	19	2.45	12.75

Fig. 13. Sequential ABMA and Dis-ABMA Master times and associated speedups.

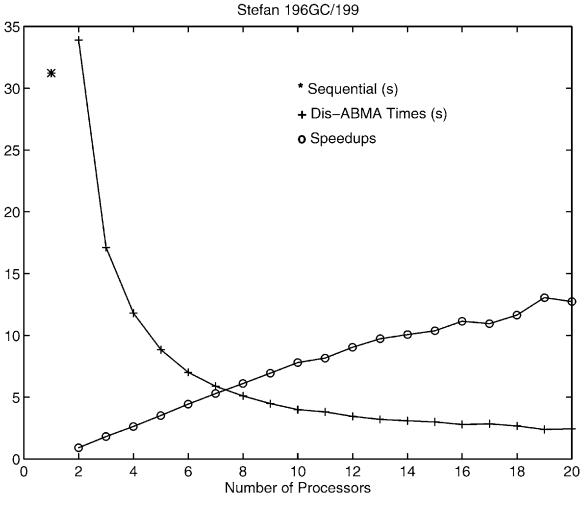

Fig. 14. Dis-ABMA Master times and speedup for Stefan 196GC/199.

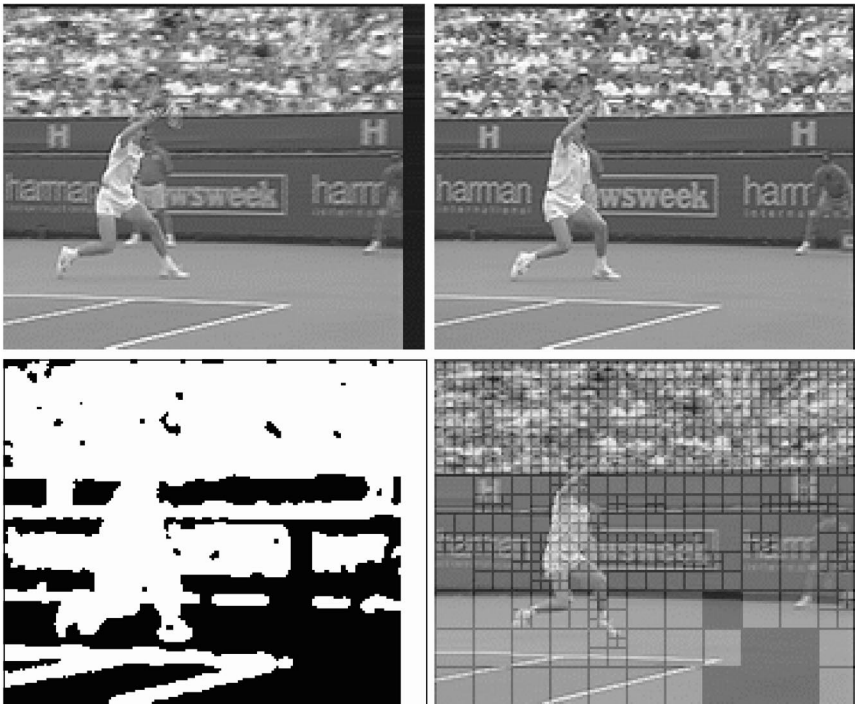

Fig. 15. "Stefan": Frame 196GC ("196GC" stands for image 196 "Globally Compensated"), Original Frame 199, Mask of changes, Dis-ABMA Compensated Frame and its multigrid.

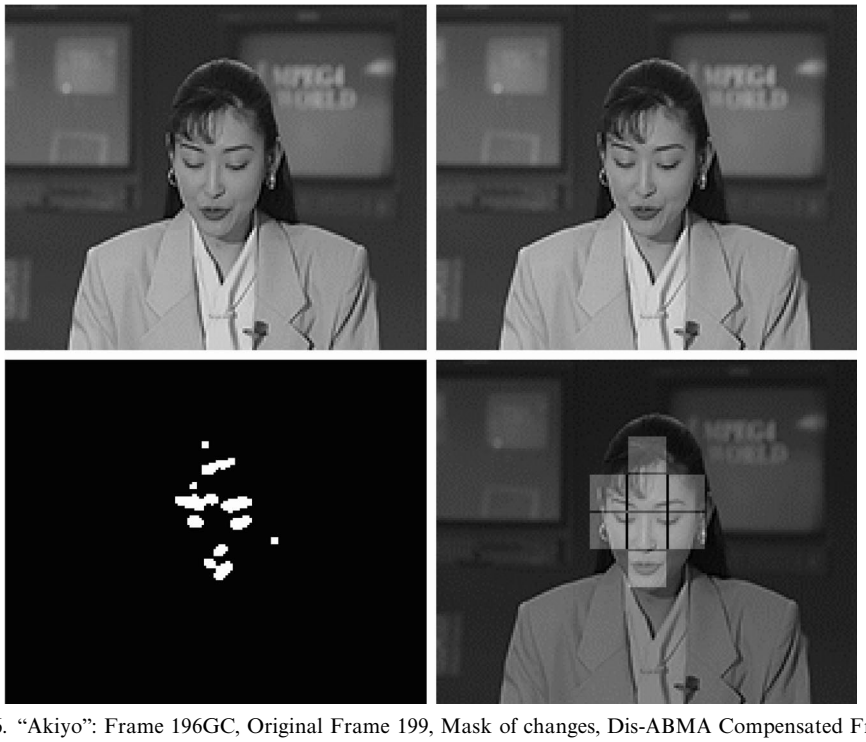

Fig. 16. "Akiyo": Frame 196GC, Original Frame 199, Mask of changes, Dis-ABMA Compensated Frame.

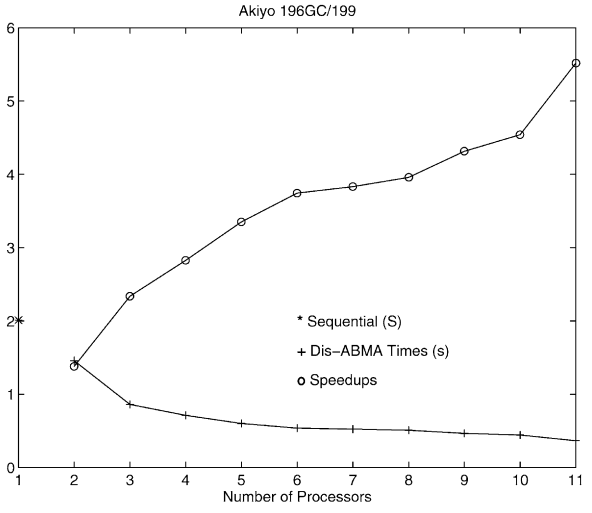

Fig. 17. Dis-ABMA Master times for the pair Akiyo 196/199.

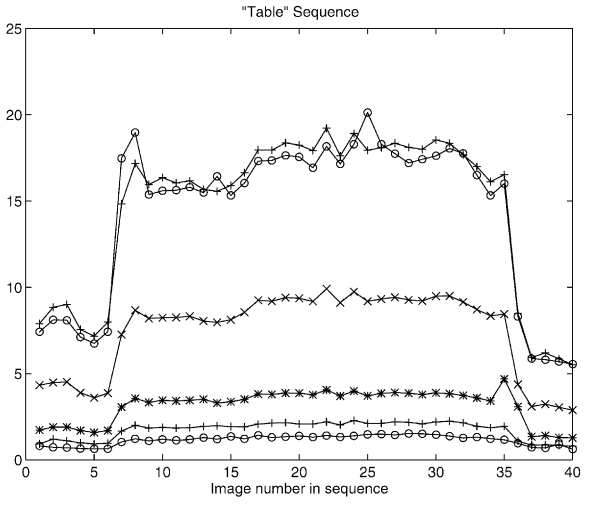

Fig. 19. Times for the images of the sequence "Table Tennis":○ Sequential, + 1 slave, × 2 slaves, ∗ 5 slaves, + 10 slaves, ○ 19 slaves.

Slave Number	1	2	3	4	5	6	7	8	9	10
1	32.15	16.05	11.37	8.17	7.27	5.88	4.77	4.56	4.05	3.60
2		16.03	11.04	8.02	7.29	5.86	5	4.51	3.93	3.62
3			11.23	7.92	7.09	5.65	4.90	4.48	4.03	3.38
4				7.49	6.95	5.61	4.87	4.44	3.94	3.36
5					6.77	5.51	4.71	4.39	3.89	3.51
6						5.07	4.80	4.42	3.94	3.41
7							4.74	4.37	3.92	3.38
8								4.25	3.42	3.14
9									4.17	3.45
10										3.53
Global exec. time (s)	36.01	17.53	12.27	8.83	7.87	6.42	5.55	5.06	4.63	5.10
Aver. Slave Idleness %	10.72	8.44	7.33	7.47	7.37	8.41	9.9	9.88	9.94	29.02

Fig. 18. Slaves computation time and average idleness.

4.3. Handling a complete sequence

Fig. 19 presents the global execution times for the first 40 images of a standard complete sequence containing various motion complexities (i.e. the MPEG-4 "Table Tennis"). This experiment is reported for the sequential ABMA and for Dis-ABMA with 1, 2, 5, 10 and 19 slaves.

One can observe that with more slaves, the graph becomes "flat". This means that Dis-ABMA can absorb the complexity of the computation process, yielding an almost constant execution time for all the images. One can thus conclude that Dis-ABMA is a possible way to ensure constant coding time for each pair of images in the sequence, whatever the motion complexity, assuming a sufficient number of slaves. Our experiments also show that, even for complex sequences, Dis-ABMA behaves very well with less than 10 slaves.

5. Conclusion

The motion estimation techniques, and in particular the Block Matching Algorithm (BMA), are used in video coders such as MPEG-1, -2, -4 coders. As the size of the blocks is critical, variable block size algorithms have been developed. A particular one is the Adaptive Block Matching Algorithm (ABMA), which is an improvment of BMA. The main disadvantage of such adaptive techniques is a slow estimation step which prevents real-time applications.

This paper presented ways to distribute the Adaptive Block Matching Algorithm. We provided a modelization of ABMA and a distributed modelization has been designed. The formal specification of this model has been expressed in terms of states of blocks, and transitions between states. This model can lead to efficient sequential implementations as well as to various distributed structures. A particular distributed architecture, using one master and several slaves, has been implemented and tested on a network of conventional workstations. Experimental results showed that the studied implementation of Dis-ABMA speeds up the coding time with a linear speedup. Owing to this distribution, the ABMA can be expected to turn real time.

This research could be continued in many directions. First of all, other distributed implementations could be developed, such as implementations on parallel computers. The load balancing technique is now hidden in the model and is quite simple. It would then be interesting to provide more elaborate load balancing techniques. Similarly, one should analyze, from the mask of change, the optimal number of slaves. Such an optimality could be defined to maintain the efficiency of the slaves, or to achieve a constant global execution time assuming that real time coding is the objective. Finally, the implementation could be improved. For instance, when the size of a block is too small, it is useless to distribute the BMA computation to slaves.

Acknowledgements

The authors thank the anonymous reviewers for helpful suggestions. This research is partly supported by the *Projet mobilisateur* "*Du Numérique au Multimédia*" (RW-EMIN 961/3467) of the Région Wallonne of Belgium.

References

[1] G.R. Andrews, Concurrent Programming Principles and Practice, The Benjamin/Cummings Publishing Company, Redwood City, 1991.

[2] A. Choudhary, R. Ponnusamy, Parallel implementation and evaluation of a motion estimation system algorithm using several data decomposition strategies, J. Parallel Distributed Comput. 14 (1992) 50–64.

[3] MPEG Video Group, MPEG-4 Video Verification Model version 3.0 ISO/IEC SC29WG11 N1277, July 1996.

[4] M. Hoetter, Differential estimation of the global motion parameters zoom and pan, Signal Processing 16 (3) (March 1989) 249–265.

[5] J.R. Jain, A.K. Jain, Displacement measurement and its application in interframe image coding, IEEE Trans. Commun. 29 (12) (December 1981) 1799–1806.

[6] Z. Kohavi, Switching & Finite Automata Theory, McGraw-Hill Computer Science Series, New York, 1970.

[7] Oak Ridge National Laboratory, PVM 3 User's guide and reference manual. Oak Ridge, Tennessee, 1993.

[8] D. Le Gall, MPEG: A video compression standard for multimedia applications, Commun. ACM 34 (4) (April 1991) 46–58.

[9] B. Macq, M.P. Queluz, B. Simon, Very low bit-rate image coding on adaptive multigrids, Signal Processing: Image Communication 7 (4–6) (November 1995) 313–331.

[10] M.L. Minsky, Computation: Finite & Infinite Machines, Prentice-Hall Series In Automatic Computation, Prentice-Hall, Englewood Cliffs, NJ, 1967.

[11] I. Pitas, Parallel Algorithms for Digital Image Processing, Computer Vision and Neural Networks, Wiley, New York, 1993.

[12] M.P. Queluz, Multiscale motion estimation and video compression, Ph.D. Thesis, Université catholique de Louvain, April 1996.

[13] M.P. Queluz, B. Macq, Signal-adapted motion compensation for video compression, ISSES, URSI, Paris, September 1992.

[14] Y.T. Tse, R.L. Baker, Global zoom/pan estimation and compensation for video compression, in: International Conference on Acoustics, Speech and Signal Processing (ICASSP), Toronto, May 1991, Vol. 4, pp. 2725–2728.

[15] G. Tziritas, C. Labit, Motion Analysis for Image Sequence Coding, Elsevier, Amsterdam, 1994.

[16] F. Vermaut, Y. Deville, B. Macq, X. Marichal, A distributed adaptive block matching algorithm: Dis-ABMA, in: European Signal Processing Conference Eusipco'98, September 1998, p. 889.