A DISTRIBUTED ADAPTIVE BLOCK MATCHING ALGORITHM : DIS-ABMA

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

Dept. of Computing Sciences and Engineering / Dept. of Electrical Engineering

Place Sainte Barbe 2, 1348 Louvain-la-Neuve, Belgium

Tel: +32 10 473150; Fax: +32 10 450345

e-mail: {fv,yde}@info.ucl.ac.be, {Macq,Marichal}@tele.ucl.ac.be

ABSTRACT

Variable block size algorithms for motion estimation, like the Adaptive Block Matching Algorithm (ABMA) [10], have been proposed to better match to "objects in motion" compared to the classical BMA [4, 2]. However, the variable block size grid derivation and the related motion estimation relies on a regularization process which implies heavy iterative and inter-dependent computation. 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 detection 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 (in C++, using the PVM platform). Experimentations show a linear speedup with respect to the number of processors.

Keywords : Image processing, motion estimation, video coding, distributed algorithms.

1 INTRODUCTION

Motion estimation (ME) algorithms are useful tools in many applications related to video signals. Such applications include video compression, video indexing, objects tracking, ... In a video coding framework, ME aims at detecting motion from one image to another. Such a motion information, eventually represented as vectors, can be used for temporal prediction of successive images [11]. The MPEG [6] coders family is based on these principles.

The Block Matching Algorithm (BMA) [4, 2] is a very popular ME algorithm because of its good compromise between result accuracy and algorithm complexity. BMA decomposes images in blocks of fixed same size. For each block of the current image one searches a similar block in the previous image.

The Adaptive Block Matching Algorithm (ABMA) [10, 9] is an improvement of BMA. In BMA, the initial size of the blocks is crucial. Large block sizes give rise to what is called "blocking artifacts". Small block sizes increase the noise sensitivity and give rise to non-zero vectors for null motions. So the idea of ABMA is

to locally adapt the size of the blocks only where it is necessary to better match 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 applied along object borders where contradictory movements arise. At an equivalent bitrate, 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 results we obtained are the following :

- An abstract specification of ABMA has been given.
- A distributed modelization of ABMA has been designed. This model can lead to different distributed versions.
- We implemented a distributed version based on the PVM platform and in terms of one master and several slaves.
- Our experimentations show a linear speedup.

In this summary paper, we focus on the main results.

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 [8, 3]). Our approach does not make any hypothesis on the underlying architecture. Although our particular implementation uses a network of conventional workstations, our model can include such massive parallel approaches.

2 BACKGROUND : THE ADAPTIVE BLOCK MATCHING ALGORITHM

The ABMA aims at adapting the size of the blocks on which the BMA is performed according to the image content. When performing a BMA, the ABMA also computes two evaluation criteria that are exposed in the following subsection. According to the value of both criteria for the block and neighbour blocks, a split decision may be taken.

2.1 Error and motion certitude

Consider two images P1 and P2 represented as matrix of pixels. Consider a block B out of P2, of size $k \times k$. The basic operation of ABMA is of course the block matching operation to find a displacement for this block from the image P1 to P2. So we are looking in P1 for a block of size $k \times k$ that is as identical as possible to B. We restrict our search into a window centered around the initial position of B (null vector). This operation is the core of the BMA.

For all positions of a block in the search window in P1 we define a Mean Absolute Error (MAE). The definition of a MAE is given by (1).

$$MAE = \frac{\sum_{i,j=1}^k |B(i,j) - B_1(i,j)|}{k^2} \quad (1)$$

with k , the size of the blocks, B_1 , the block in P1 and B the block in P2.

Inside the search window we can find a number of position for a block of size k equal to $(2 * dh + 1) * (2 * dv + 1)$, with dh the distance for horizontal search and dv the distance for vertical search. As the complexity of one computing of a MAE is $\mathcal{O}(k^2)$, the temporal complexity for BMA on one block is $\mathcal{O}(k^2 * dh * dv)$.

The detected motion can sometimes be noise-driven and thereafter not represent the movement really executed by a block between the two images. The notion of *the motion certitude* (MC) introduced in ABMA aims at representing the trust we can have in the movement found by BMA. MC is defined by (2).

$$MC = \frac{\sum_{i=1}^{N_s} MAE_i}{N_s} - MAE_{min} \quad (2)$$

with N_s , the number of positions for the block in the search window $(2 * dh + 1) * (2 * dv + 1)$, and MAE_{min} the minimal MAE among the MAE's computed for all the positions.

More information on MAE and MC can be found in [10].

2.2 Adaptive Block Matching Algorithm (ABMA)

ABMA is composed of 2 steps each working on a pair of images.

1. A mask of changes detection : To reduce the amount of computations of the next step, ABMA computes a mask of changes.
2. A motion detection, using BMA among others.

Motion detection is the main part of ABMA. It gives as result a multigrid, and for each block of the multigrid a motion vector.

As motion detection is the most expensive part of the algorithm, we decided to distribute it. The original ABMA will not be described here. The reader is referred to [5, 9, 10] for more information. In the next sections,

Dis-ABMA, a distributed version of ABMA will be presented. Although it is a distributed modelization, this is also a useful abstraction of the basic procedural description of the original ABMA.

3 MODEL OF DISTRIBUTION

3.1 States and transitions of blocks

The main “object” in this algorithm is *the block*. As we are defining a model of distribution, it is important to distinguish the various states of a block during the ABMA, to precise the possible transitions, and how synchronization can be performed. This will modelize the “life” of a block. We therefore propose a *finite state automaton*. [1]

Figure 1: State Diagram

The initial state results from the change detection step. Some blocks are to be treated and some are already detected as having a null motion. For the treatable blocks, no information is known for the block. The initial state for these blocks is called UNKNOWN. The other blocks are in a state called DONE because their treatment is finished.

The first operation that must be applied to an UNKNOWN block is the BMA operation. It then comes in the KNOWN state. When a block is in the KNOWN state as well as its neighbours, it should update its motion vector according to the information from these neighbours. This update is called *Motion certitude treatment*. The objective of this treatment is to try to make uniform the motion vectors between adjacent blocks. When such a treatment is performed, the block moves to the PROPAGATED state.

When a block is in the PROPAGATED state as well as its neighbours, it can then decide whether its motion vector is accepted, or if this should be further refined. In this later case, the block is split into four new smaller

blocks, while the original block becomes in the SPLIT state. As some filtering is performed during this decision, this transition is called *Filtering and decision*. The initial state of these smaller blocks is the UNKNOWN state. Of course, small blocks of a predefined limited size will not be further split.

3.2 Distribution

This model gives us the possibility to create a sequential version but also different distributed structures. We chose a simple one. As BMA is the most expensive computation, we decided to distribute this step between slaves. The slave can thus perform only one action : BMA. But several BMA are simultaneously computed by several slaves. We enrich our state diagram with the new state WAITING. During the BMA operation, a block will be in the WAITING state.

A master is used to distribute the different blocks, to receive the results of computation of the slaves and to perform motion certitude treatment and filtering.

In the sequential version, these two last steps are made successively on all the blocks only when BMA is done for all blocks. In the distributed version, it is achieved *as soon as possible*. One should therefore know what are the conditions for a transition. This will be formalized in the next section.

Figure 2: Master-Slaves structure

4 DIS-ABMA

Our goal is to make a block an independent computation unit with relations to neighbour blocks.

4.1 Data structures

We first suppose that the master and all the slaves possess the two images P2 and P1. It is the only information the slaves possess.

For the master, four new structures are needed. The first one is the entire multigrid, which is the final result of ABMA. The other three are described hereafter.

WWL =	Waiting Works List, list of blocks to be treated by a slave.
WSL =	Waiting Slaves List, list of free slaves.
block2slave =	list of pairs <slave,block> indicating which block is possessed by a slave.
Invariant $\forall b : \text{block} \quad b \in WWL \Leftrightarrow b.State = UNKNOWN$	

The data structure of a block b is the following.

$b.size =$	The size of block b
$b.State =$	The current state of bloc b
$b.MAE_s =$	The MAE of bloc b at state $s \in \{KNOWN, PROPAGATED, DONE\}$
$b.V_s =$	The Motion Vector of bloc b at state $s \in \{KNOWN, PROPAGATED, DONE\}$
$b.MC_s =$	The Motion Certitude of bloc b at state $s \in \{KNOWN\}$

Other information such as the block position and a list of its neighbours is also included in the data structure.

4.2 Formal specification of transitions

This is the central part of the modelization. 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 specify when the transition can be triggered. In other words, this characterizes conditions, on the environment of a blocks, that must be fulfilled before the transition can be performed.

As soon as a slave is free, a block in UNKNOWN state can be sent to that slave for BMA completion. The block falls in WAITING state. When the slave returns the results, the block may go in KNOWN state.

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

In order to treat the motion certitude, a block needs the motion vector of four specific neighbours. We call the neighbours of *4-connectivity* the four neighbours North, East, South, West. The eight classical neighbours are called neighbours of *8-connectivity*. So when the neighbours of 4-connectivity are in the KNOWN

state, the certitude of the block can be treated. Some of the neighbour blocks may have already used their certitude and so already be in PROPAGATED state. In this case, one has to use the vector value before motion certitude treatment. As we are not considering a sequential global execution, no hypothesis can be made on the states of the neighbours nor even on their size. The triggering information must thus be enhanced to handle such cases. When a neighbour has a bigger size, it must be in the DONE state (otherwise, the block should wait for a possible new smaller neighbour resulting from the splitting of this large neighbour).

Similar reasonings can be made for the filtering and decision steps.

MOTION CERTITUDE TREATMENT

$\forall b : \text{block}$

pre : $b.State = \text{KNOWN}$

post : $b.State = \text{PROPAGATED}$
 $\wedge b.V_P = \text{vector value after motion}$
 $\text{certitude treatment}$
 $\wedge b.MAE_P = \text{associated MAE}$

trigger : $\forall n : \text{neighbour of 4-connectivity}$
 $[n.size = b.size \wedge (n.State = \text{KNOWN}$
 $\vee n.State = \text{PROPAGATED})]$
 $\vee [n.size > b.size \wedge n.State = \text{DONE}]$

FILTERING AND DECISION

$\forall b : \text{block}$

pre : $b.State = \text{PROPAGATED}$

post : $(b.State = \text{DONE} \wedge b.V_D = \text{final vector value}$
 $\text{after filtering} \wedge b.MAE_D = \text{associated MAE})$
 $\vee (b.State = \text{SPLIT}$
 $\wedge WWL = WWL \cup \{b_1, b_2, b_3, b_4\})$
 $\text{with } b_1, b_2, b_3, b_4 \text{ four new smaller blocks.}$

trigger : $\forall n : \text{neighbour of 8-connectivity}$
 $[n.size = b.size \wedge (n.State = \text{PROPAGATED}$
 $\vee n.State = \text{DONE} \vee n.State = \text{SPLIT})]$
 $\vee [n.size > b.size \wedge n.State = \text{DONE}]$

5 IMPLEMENTATION AND EXPERIMENTAL RESULTS

Our model for Dis-ABMA has been implemented (in C++) using a network of conventional workstations. The PVM platform [7] for distribution has been used.

We realized several tests on an Ethernet linking 10 SUN computers. Our parameters were the number of slaves and the complexity of the motion existing between our two images. Generally speaking, we observed a linear speedup with an efficiency of 50 %. This is primarily due to the poor performances of an Ethernet network and PVM on it. Figure 3 illustrates one of the benchmarks.

6 CONCLUSION

This paper presented ways to distribute a motion estimation algorithm that improves the BMA. A distributed modelization of the so-called ABMA has been proposed,

Figure 3: Computation Times and Linear Speedup

and a distributed implementation has been developed. It speeds up the coding time with a linear speedup. Thanks to this distribution, the ABMA is expected to turn realtime.

References

- [1] Andrews G.R. *Concurrent programming principles and practice*. The Benjamin/Cummings Publishing Company, Redwood City, 1991.
- [2] Chiou Shann Fuh, Petros Magaros. Affine models for image matching. Technical report, Division of Applied Science, Harvard University, Cambridge, 1991.
- [3] Choudhary A., Ponnusamy R. . Parallel implementation and evaluation of a motion estimation system algorithm using several data decomposition strategies. In *Journal of Parallel and Distributed Computing*, volume 14, pages 50–64, 1992.
- [4] Jain J.R., Jain A.K. Displacement measurement and its application in interframe image coding. *Itcomm*, 29(12):1799–1806, Dec 1981.
- [5] Macq B., Queluz M.P., Simon, B. Very low bit-rate image coding on adaptive multigrids. *Signal-Processing : Image Communication*, 7(4-6):313–331, Nov 1995.
- [6] MPEG Video Group. MPEG-4 Video Verification Model version 3.0 ISO/IEC SC29WG11 N1277. , Jul 1996.
- [7] Oak Ridge National Laboratory. *PVM 3 User's guide and reference manual*. Oak Ridge, Tennessee, 1993.
- [8] Pitas I. *Parallel Algorithms for Digital Image Processing, Computer Vision and Neural Networks*. John Wiley & Sons Ltd, 1993.
- [9] Queluz M.P. *Multiscale Motion Estimation and Video Compression*. PhD thesis, UCL, Apr 1996.
- [10] Queluz M.P., Macq B. Signal-adapted motion compensation for video compression. Paris, Sep 1992. ISSes, URSI.
- [11] Tziritas G., Labit C. *Motion Analysis for Image Sequence Coding*. Elsevier, Amsterdam, 1994.