AMERICAN UNIVERSITY OF BEIRUT

ON PARALLEL, DISTRIBUTED, AND HYBRID EVOLUTIONARY ALGORITHMS FOR BLOCK MOTION ESTIMATION

by
MANAL KHALIL JALLOUL

A dissertation
submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
to the Department of Electrical and Computer Engineering
of the Faculty of Engineering and Architecture
at the American University of Beirut

Beirut, Lebanon
April 2016

AMERICAN UNIVERSITY OF BEIRUT

ON PARALLEL, DISTRIBUTED, AND HYBRID EVOLUTIONARY ALGORITHMS FOR BLOCK MOTION ESTIMATION

by

MANAL KHALIL JALLOUL

Approved by:

Dr. Ayman Kayssi, Professor Electrical and Computer Engineering	Chair of the Committee
Dr. Mohamad Adnan Al-Alaoui, Professor Electrical and Computer Engineering	Advisor
Dr. Ali Chehab, Professor Electrical and Computer Engineering	Member of Committee
Dr. Haitham Akkary, Professor Electrical and Computer Engineering	Member of Committee

Dr. Daniel Asmar, Associate Professor
Mechanical Engineering

Member of Committee

Dr. Ali Sayed, Professor
Electrical Engineering, University of California, Los Angeles, USA

M. A. Al-Akaidi (On Behalf of Prof. Sayed)
Member of Committee

Dr. Haidar Harmanani, Professor
Computer Science, Lebanese American University, Lebanon

Member of Committee

Dr. Maha El Choubassi, Senior Research Scientist
Intel Corporation, USA

M. A. Al-Akaidi (On Behalf of Dr. El Choubassi)
Member of Committee

Dr. Rony Fezli, Soft Architect
Intel Corporation, USA

M. A. Al-Akaidi (On Behalf of Dr. Fezli)
Member of Committee

Date of dissertation defense: April 22, 2016

AMERICAN UNIVERSITY OF BEIRUT

DISSERTATION RELEASE FORM

Student Name: Jalloul (Last) Manal (First) Khalil (Middle)

○ Master's Thesis ○ Master's Project ● Doctoral Dissertation

☐ I authorize the American University of Beirut to: (a) reproduce hard or electronic copies of my thesis, dissertation, or project; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes.

☐ I authorize the American University of Beirut, **three years after the date of submitting my dissertation,** to: (a) reproduce hard or electronic copies of it; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes.

_____

Signature Date

ACKNOWLEDGMENTS

First and foremost, I would like thank Al-Mighty God, whose showers of blessings have made me who I am today.

The deepest gratitude goes towards my parents and family for their love and support throughout my life. Thank you for encouraging me in all my pursuits and inspiring me to follow my dreams.

My greatest appreciation goes to Prof. Mohamad Adnan Al-Alaoui. He has been an advisor, a mentor and a friend. His care and guidance made achieving this work possible. His zeal towards the thesis progress considerably improved the quality of this work.

I would like to extend my thanks to my committee members for their constructive feedback that helped enhancing the quality of this work. I would like to acknowledge Prof. Walid Saad. He showed me support and guidance and provided constructive feedback in the game theory part of the thesis. He was ready to answer any consultation or inquiry at any point in time.

I am also greatly thankful to the American University of Beirut Research Board (URB fund) for the financial support that made this work possible.

I also like to acknowledge NVIDIA for their support in starting the GPU academic center in AUB. This was the main motivation for me to delve into the area of parallel and GPU programming.

I'm also greatly thankful to Dr. Mohammed Baydoun. His technical advice in GPUs and CUDA programming has been very beneficial.

I would like to extend my thanks to all my friends and colleagues at AUB for their constant support. Their encouraging words have kept me going when coffee had lost its stimulating effect. My deepest gratitude goes to Dr. Ahmad El Hajj, Dr. Lina Al-Kanj, Dr. Dima El-Khalil, Dr. Lise Safately, Dr. Rawad Assi, Dr. Jihad Fahs, and Dr. Hilal El Misilmani. Thank you for your friendship and lovely memories.

Finally, I would to wholeheartedly thank my second half, my husband Dany, for his overwhelming love, encouragement, and constant support. Last but not least, words fail to express how fortunate and grateful I am to have two lovely daughters, Noor and Sama. You are the essence of my life, the drive that makes me want to accomplish more. I dedicate this thesis to you.

AN ABSTRACT OF THE THESIS OF

Manal Khalil Jalloul for Doctor of Philosophy
Major: Electrical and Computer Engineering

Title: On Parallel, Distributed, and Hybrid Evolutionary Algorithms for Block Motion Estimation

Motion estimation (ME) is a common tool used in all video coding standards. Fast and accurate algorithms are needed to target the real-time processing requirements of emerging applications. On the other hand, in the hardware industry, there is great emphasis on High Performance Computing (HPC) which is characterized by a shift to multi and many core systems. The programming community has to embrace the new parallelism in order to take advantage of the performance gains offered by the new technology. The block motion estimation (BME) problem is classified as non-convex since the objective function is multimodal. Existing fast block matching methods suffer from poor accuracy and are susceptible to being trapped into local optima on the error surface. The collective intelligence enabled by the particle swarm optimization (PSO) technique, however, was found effective in alleviating the local optima problem. Belonging to the category of evolutionary algorithms, PSO is capable of handling non-differentiable, discontinuous and multimodal objective functions. To this end, in this dissertation, several efficient and parallel ME algorithms based on PSO are proposed. Several levels of parallelisms are introduced into the ME process. First, parallelism between the macroblocks (MBs) of the frame is achieved through a novel cooperative ME scheme based on a multi-swarm PSO model that performs ME in a cooperative manner concurrently for all the MBs in the frame. Several strategies are incorporated into the dynamics of the PSO algorithm to improve its motion estimation accuracy and enhance its convergence speed including a novel initialization scheme, a fitness function history preservation algorithm, and a dynamically varied maximum velocity. The multi-core and GPU implementations of the proposed framework showed that the speedup provided is scalable with the video resolution. Second, parallelism is introduced within the MB through two different approaches based on distributed multi-agents systems. The problem of BME is first cast in a non-cooperative game-theoretic setting and formulated as a potential game. To solve the game, distributed sequential and simultaneous algorithms based on game-theoretic Best Response Dynamics (BRD) and PSO are presented. Parallelism within the MB is also tackled using concepts from diffusion adaptation. The distributed optimization of BME is formulated based on diffusion protocols and a modified dynamic diffusion-based PSO algorithm is proposed to solve it. Performance evaluations and multi-core implementations of these algorithms demonstrate the merits of the presented schemes. Moreover, this dissertation also targets ME in high resolution video where a hybrid PSO-genetic algorithm is proposed and evaluated.

CONTENTS

ILLUSTRATIONS

Figure Page

TABLES

CHAPTER 1

INTRODUCTION

In the 21st century, the modern society has made itself into the global information age in which images and videos can be found everywhere in people's daily life. Nearly over 2.6 million hours of video are uploaded to YouTube each month [1]. Also, the resolution of video has grown dramatically from 100x100 in the 1960s to around 8192x4320 for video nowadays. As a result, the size of raw digital source data can be so tremendous that enormous resources are required for storage and transmission. For example, the size of a 150-minute color movie with 30 frames per second and 720x480 resolution is as large as 280 GB without compression, not to mention the situation when the movie needs to be transmitted through the Internet whose bandwidth can be lower than 10 Mbit/s. In light of this, digital video compression technology is a necessity even though computer power, storage, and the network bandwidth have increased significantly.

1.1 Video Coding

Today, video coding has become the central technology in a wide range of applications, as shown in Fig. 1.1. Some of these include digital TV, DVD, Internet streaming video, video conferencing, distance learning, surveillance, and security.

Figure 1.1 Applications of Video Coding

Video coding standards have evolved primarily through the development of the well-known ITU-T and ISO/IEC standards. The ITU-T produced H.261 [3] and H.263 [4], ISO/IEC produced MPEG-1 (ISO/IEC JTCl/SC29/WG11, December 1991) and MPEG-4 Visual, and the two organizations jointly produced the H.262/MPEG-2 [2] Video and H.264/MPEG-4 [5] AVC standards. These two organizations have been working together in a partnership known as the Joint Collaborative Team on Video Coding (JCT-VC) to produce the HEVC, the High Efficiency Video Coding standard, which is the most recent video coding standard. The first edition of the HEVC standard was finalized in January 2013[6]. Fig. 1.2 shows the chronology of video coding standards [7].

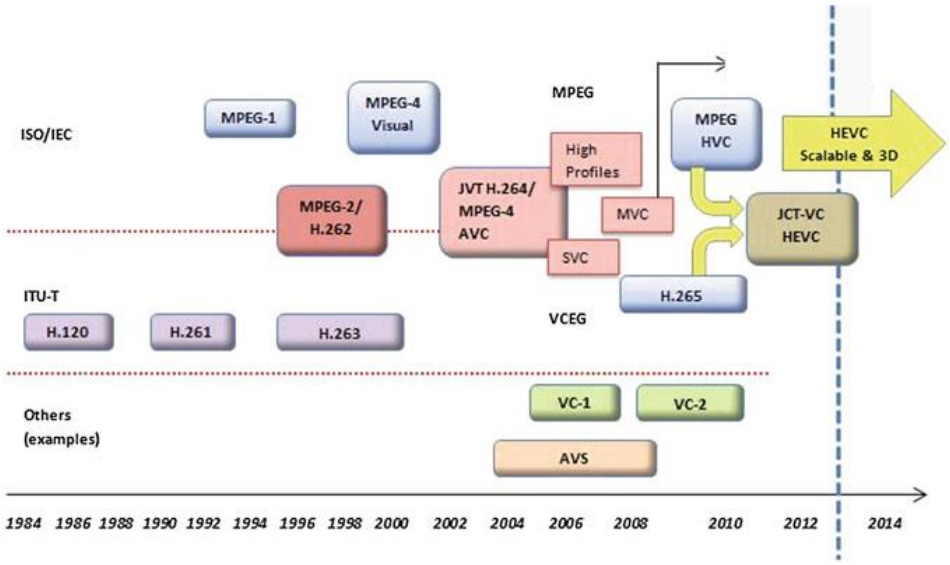


Figure 1.2 Chronology of Video Coding Standards [7]

Inter-prediction motion estimation is a common tool used in all video coding standards. The H.264/MPEG-4 AVC video coding standard and the recent HEVC standard employ the same hybrid approach to achieve high compression performance.

Fig. 1.3 shows a block diagram of a generic video encoder [7]. Motion-estimation is used to find motion of macro-blocks using motion vectors to reduce temporal redundancies among input frames. Later, transform (mostly Discrete Cosine Transform: DCT) is performed on the motion-compensated prediction difference frames for de-correlation of prediction error. The prediction error is later quantized as per input bit-rate requirements. The quantized DCT coefficients, motion vectors, and side information are entropy coded using variable length codes (VLC's). The reconstruction path in encoder consists of inverse transform, quantization, loop filter and motion compensation to mimic operation on decoder side.

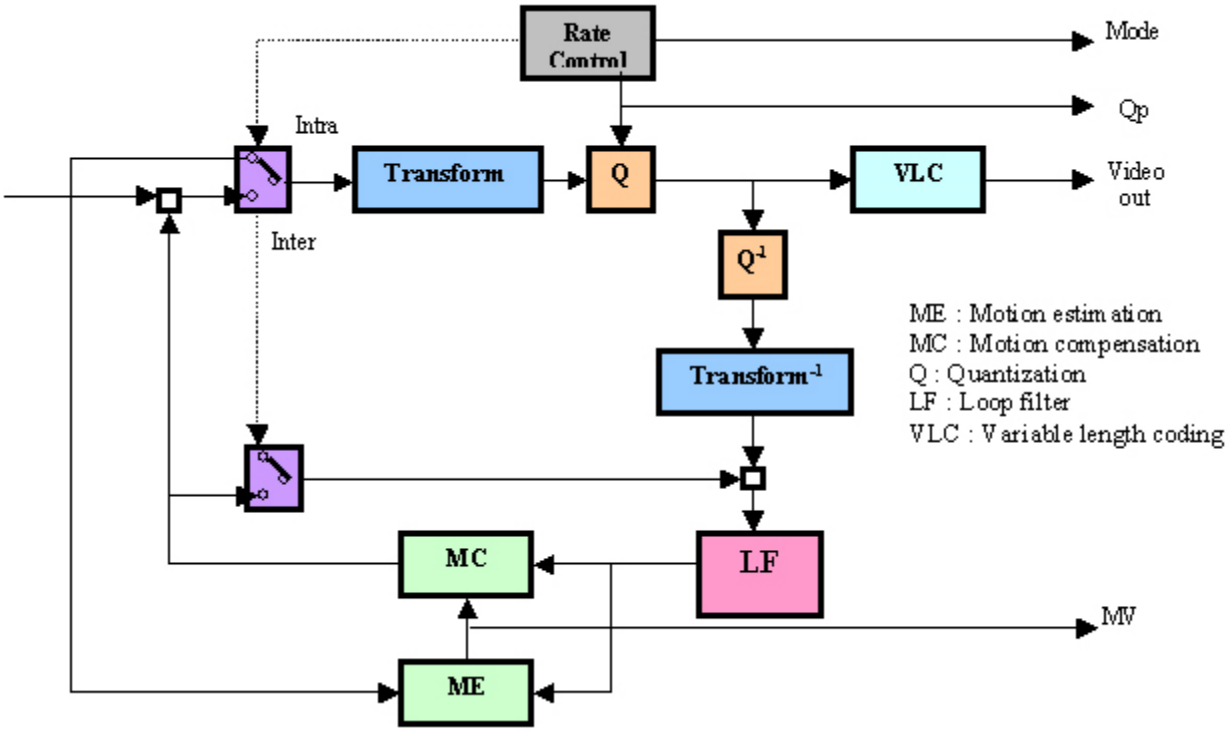


Figure 1.3 Generic Video Encoder's Block Diagram

Inter-prediction motion estimation is considered the most computationally intensive feature of the coding process. It represents about 80% of the total computational complexity of current video coders [8].

1.2 Motion Estimation

Motion Estimation (ME) is an important part of any video coding system since it can achieve significant compression by exploiting the temporal redundancy that commonly exists in a video sequence. There exist two basic approaches to motion estimation which are pixel-based motion estimation, that include parametric based models [9], optical flow [10], and pel-recursive techniques [11], and block-based motion estimation. The pixel-based motion estimation approach

seeks to determine motion vectors for every pixel in the image. This works on the fundamental assumption of brightness constancy, that is, the intensity of a pixel remains constant when it is displaced. However, no unique match for a pixel in the reference frame is found in the direction normal to the intensity gradient. It is for this reason that an additional constraint is also introduced in terms of the smoothness of velocity (or displacement) vectors in the neighborhood. The smoothness constraint makes the algorithm interactive and requires excessively large computation time, making it unsuitable for practical and real-time implementation [12].An alternative and faster approach is the block-based motion estimation (BM). In this method, the candidate frame is divided into non-overlapping blocks. It is assumed that all the pixels within a block have the same motion activity and one motion vector is estimated for each block.BM seems to be the most popular technique due to its effectiveness and simplicity for both software and hardware implementations [13]. In order to reduce the computational complexity in ME, many BM algorithms have been proposed and employed at implementations for several video compression standards.

The effectiveness of compression techniques that use block-based motion compensation depends on the extent to which the following assumptions hold:

- The illumination is uniform along motion trajectories.
- The problems due to uncovered areas are neglected.

For the first assumption it neglects the problem of illumination change over time, which includes optical flow but does not correspond to any motion. The second assumption refers to the uncovered background problem. Basically, for the area of an uncovered background in the reference frame, no optical flow can be found in the reference frame. Although these assumptions do not always hold for all real-world video sequences, they continue to be used as the basis of

many motion estimation techniques.

1.3 Principle of the Block Matching Algorithm

Figure 1.4 illustrates the process of the block-matching algorithm. In a typical BM algorithm, the current frame of an image sequence I^t is divided into non-overlapping macroblocks (MB) of $N \times N$ pixels, each of which consists of luminance and chrominance blocks. Usually, for coding efficiency, motion estimation is performed only on the luminance block. For each template luminance block in the current frame, the best matched block within a search window (S) of size $(2\mathrm{W} + 1) \times (2\mathrm{W} + 1)$ in the previous frame $I^{t-1}$is determined, where Wis the maximum allowed displacement. The position difference between a template block in the current frame and the best matched block in the previous frame is called the motion vector (MV). In a typical inter-frame coder, the input frame is subtracted from the prediction of the reference frame. Consequently, the motion vector and the resulting error can be transmitted instead of the original luminance block; thus inter-frame redundancy is removed and data compression is achieved. At receiver end, the decoder builds the frame difference signal from the received data and adds it to the reconstructed reference frames. The summation gives an exact replica of the current frame. The better the prediction the smaller the error signal and hence the transmission bit rate.

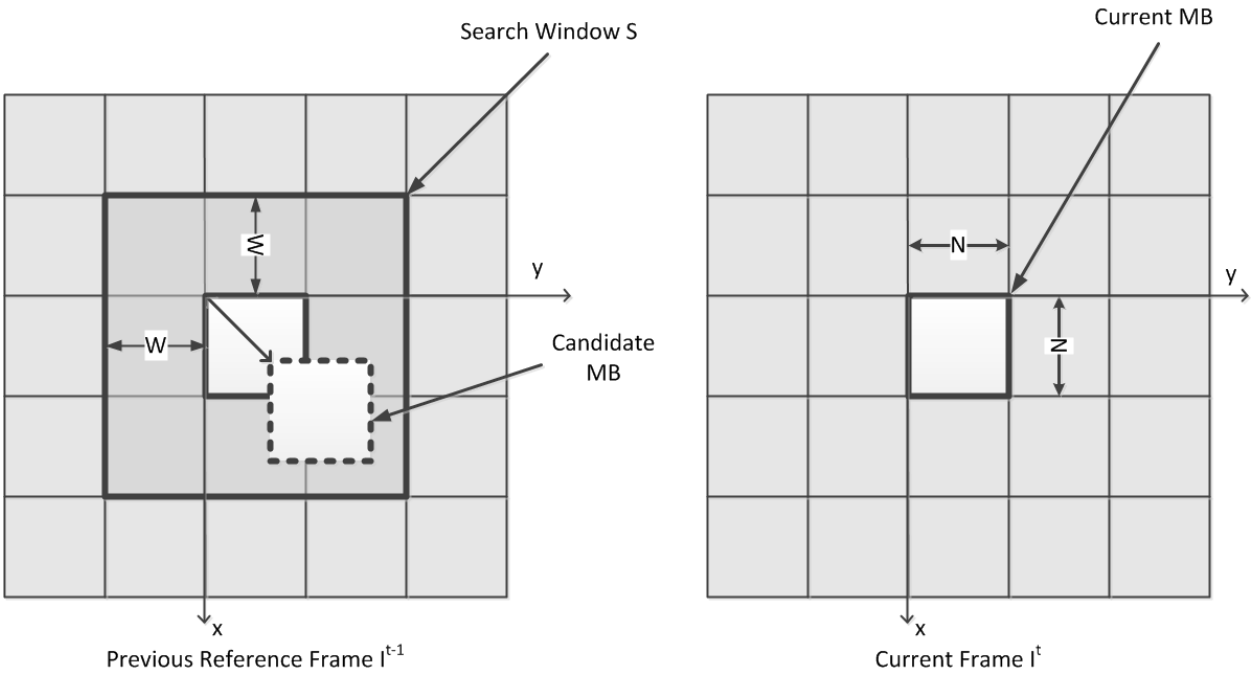


Figure 1.4 Block motion estimation

The most well-known criterion for BM algorithms is the sum of absolute differences (SAD). It is defined in Eq. (1.1) considering a template MB at position (*x*, *y*) in the current frame and the candidate MB at position $(x + \hat{u}, y + \hat{v})$ in the previous frameI^{t-1}:

$$SAD(\hat{u}, \hat{v}) = \sum_{j=0}^{N-1} \sum_{i=0}^{N-1} |g_t(x + i, y + j) - g_{t-1}(x + \hat{u} + i, y + \hat{v} + j)|, \quad (1.1)$$

where $g_t(.)$ is the gray value of a pixel in the current frame I^t and $g_{t-1}(.)$ is the gray level of a pixel in the previous frameI^{t-1}. Therefore, the MV $w =$ (u, v)is defined as follows:

$$w = (u, v) = arg_{(u,v)\in S} \min SAD(\hat{u}, \hat{v}), \quad (1.2)$$

where

$$S = \{(\hat{u}, \hat{v}) | -W \leq \hat{u}, \hat{v} \leq W \; and \; (x + \hat{u}, y + \hat{v}) is \; a \; valid \; pixel \; position \; in \; I^{t-1}\}.$$

1.3.1 The Exhaustive Search Algorithm

In the context of BM algorithms, the Exhaustive Search (ES) algorithm is the most robust

and accurate method to find the MV. It tests all possible candidate blocks from I^{t-1}within the search area to find the block with the minimum SAD. For the maximum displacement of *W*, the FSA requires $(2W+1)^2$search points. For instance, if the maximum displacement W is ± 7, the total search-points are 225. Each SAD calculation requires $2N^2$additions and the total number of additions for the ES to match a 16 × 16 block is 130,560. Such computational requirement makes the application of ES difficult for real time tasks.

1.3.2 Fast Block Matching Algorithms

Many fast search algorithms have been proposed to reduce the computational complexity of ES while retaining similar prediction quality. All of them make use of the quadrant monotonic model [14]. The quadrant monotonic model assumes that the value of the distortion function increases as the distance from the point of minimum distortion increases. Therefore, not only the candidate blocks close to the optimal block better match than those far from it, but also the value of the distortion function is a function of the distance from the optimal position. Thus, the quadrant monotonic assumption is a special case of the principle of locality. The quadrant monotonic assumption allows for the development of suboptimal algorithms that examine only some of the candidate blocks in the search area. In addition, they use the values of the distortion function to guide the search toward a good match. As the entire candidate blocks are not examined, the match found might not be the best available.

Existing fast BM algorithms were designed using the following three techniques: (1) using a fixed pattern: the search operation is conducted over a fixed subset of the total search window. The Three Step Search (TSS) [15], the New Three Step Search (NTSS) [16], the Simple and Efficient TSS (SES) [17], the Four Step Search (4SS) [18], the Diamond Search (DS) [19],the

cross-diamond search (CDS) method [20], and the Hexagon-based search [21] all represent some of its well-known examples. Although such approaches have been algorithmically considered as the fastest, they are not able to eventually match the dynamic motion content, sometimes delivering false motion vectors (image distortions). These algorithms reduce the computational complexity with negligible loss of image quality only when the motions matched the pattern well; otherwise, the image quality will decrease.(2) Reducing the search points: the algorithm chooses as search points only those locations that iteratively minimize the error-function (SAD values). This category includes the Adaptive Rood Pattern Search (ARPS) [22], the Fast Block Matching Using Prediction (FBMAUPR) [23], the Block-based Gradient Descent Search (BBGD) [24] and the Neighborhood Elimination algorithm (NE) [25]. Such approaches assume that the error-function behaves monotonically, holding well for slow-moving sequences but failing for other kind of movements in video sequences [26], making the algorithm prone to get trapped into local minima. (3) Decreasing the computational overhead for every search point: the matching cost (SAD operation) is replaced by a partial or a simplified version that features less complexity. The New pixel-Decimation (ND) [27] and the Successive Elimination Algorithm [28] assume that all pixels within each block, move by the same finite distance and a good estimate of the motion can be obtained through only a fraction of the pixel pool. However, since only a fraction of pixels enters into the matching computation, the use of such regular sub-sampling techniques can seriously affect the accuracy of the detection of motion vectors due to noise or illumination changes. Another popular group of BM algorithms employ spatio-temporal correlation by using neighboring blocks in the spatial and temporal domain in order to predict MVs. The main advantage of such algorithms is that they alleviate the local minimum problem to some extent as the new initial or predicted search center is usually closer to the global minimum and therefore the

chance of getting trapped in a local minimum decreases. This idea has been incorporated by many fast-block motion estimation algorithms such as the Unsymmetrical Multi-Hexagon-grid search (UMHexagonS) [29]. However, the information delivered by the neighboring blocks occasionally conduces to false initial search points producing distorted motion vectors. Such problem is typically caused by the movement of very small objects contained in the image sequences [30].The UMHexagonS [29] algorithm attempt to use many search patterns, has achieved both fast speed and good rate-distortion performance. As a result, it was adopted in H.264/AVC reference software JM. Although uneven search patterns are used to meet the assumption that motion is more horizontal than vertical, it cannot adaptively choose the intensive search area for irregular motions. To tackle this drawback, Predictive Intensive Direction Searching (PIDS) algorithm [31] was developed. In PIDS, the correlation of predicted MV and optimal MV are studied. On the basis of MV prediction information, the area with high correlation is intensively searched, while other areas are coarsely searched. PIDS successfully speeds up the process compared to UMHexagonS. However, this algorithm still searches each direction exhaustively, which may cause searching resource waste. In [32], a novel Predictive Priority Region Search (PPRS) algorithm that performs adaptively search indirection and locality regions was proposed. In this proposed algorithm, the search window is divided by 8 direction and several octagon grids. These regions are then selectively searched by exploiting the MV correlation characteristics of the previous encoded frame. Other FME algorithms proposed in the literature include Motion adaptive search (MAS) [33] which utilize the motion activity information to adjust the search strategy, Variable Step Search (VSS) algorithm [34] which employs correlation between neighboring motion vectors to determine motion search range, and the Multi-Path Search (MPS) algorithm [35] in which all the eight neighbors around the origin of the search window are used to find candidate points. In

addition to the above, several high efficiency algorithms were presented in the literature for ME. These algorithms significantly reduce the number of checking points examined while retaining the video quality. These techniques accomplish this by initially considering several highly likely predictors, introducing very reliable early-stopping criteria to terminate the search at any checking point, and using very efficient checking patterns for optimizing and improving the search even further. These algorithms include the Motion Vector Field Adaptive Search Technique (MVFAST) [36], the Predictive Motion Vector Field Adaptive Search Technique (PMVFAST) [38], the Advanced Predictive Diamond Zonal Search (APDZS) [38], and the Enhanced Predictive Zonal Search (EPZS) [39].

1.3.3 Evolutionary Algorithms for Block Motion Estimation

Block matching motion estimation can be formulated into an optimization problem where one searches for the optimal matching block within a search region which minimizes a certain block distortion measure (BDM), which is usually taken as the sum of absolute difference. Such a problem is classified as non-convex since the objective function is multimodal and has many local minima. The above fast block matching methods suffer from poor accuracy since they dictate that only a very small fraction of the entire set of candidate blocks be examined, thereby making the search susceptible to being trapped into local optima on the error surface. The underlying theory of these search engines comes from the idea that the block distortion measure reduces monotonously when search points move from the farthest point toward the optimal point. In practice, applications do not always completely obey the monotonous rule. Therefore, these fast search engines are easily trapped into the local optimal solutions and miss the global optimal solution. In order to escape from the problem of local minima, several approaches were recently presented in the

literature to use modern global optimization algorithms to solve the problem of motion estimation. In [40, 41], the Genetic Algorithm (GA) has been considered for motion estimation. The proposed algorithms, however, tend to be complex and suffer from a high computational burden. In [42], the Simulated Annealing (SA) concept is employed to control the searching process and to adaptively choose the intensive search region. In addition, artificial bee colony optimization (ABC) [43] and differential evolution (DE) [44] were also proposed for motion estimation.

Recently, there have been some attempts in the literature to apply Particle Swarm Optimization (PSO) to solve the problem of ME [45-52]. The PSO-based motion estimation methods introduced in [45-49] either have higher computational complexity [45] or have lower estimation accuracy [46-48, 51] than several existing fast search methods, such as the three-step search (TSS) and diamond search(DS) method. For example, in [47], a method called zero-motion pre-judgment was applied to PSO based motion estimation to reduce the computational complexity. However, this fast method caused a significant degradation in motion estimation accuracy. In [45], a parallel PSO method was applied to block-based motion estimation to reduce the computational cost, albeit at the cost of substantially lowered estimation accuracy. Moreover, the simulation results reported in [48] were too limited to demonstrate the suitability and effectiveness of the PSO method for block-based motion estimation. In [51], a pattern-based PSO approach was proposed for block motion estimation. To speed up the conventional PSO, the algorithm presented in [51] selects the initial position of the particles in a fixed pattern rather than randomly as in the conventional PSO scheme. PSO particles are initialized in a square or a diamond pattern around the center. In [52], the standard PSO algorithm was modified to meet the stringent constraint of low computational complexity while maintaining high motion estimation accuracy. This is done by employing several strategies to speed up the motion estimation process

and gave high motion estimation accuracy as compared to 4SS, DS, and CDS. This algorithm tries to improve the speed of convergence of the PSO iterations by choosing, as initial positions of the particles, the MVs of adjacent causal blocks in the frame as well as the (0,0) MV.

1.4 Motivation

Motion estimation is a common tool used in all video coding standards. Fast and accurate algorithms are needed to target the real-time processing requirements of emerging applications. As was shown in section, existing techniques have several drawbacks. The main focus of this PhD thesis work is to develop efficient motion estimation algorithms that overcome the drawbacks of existing approaches. The motivation for this work can be summarized by the following points:

1.4.1 Fast and Accurate BM algorithm

Designing block matching algorithms that are both fast and accurate presents a challenge. We have seen that existing fast BM algorithms are susceptible to being trapped into local optima on the error surface. The collective intelligence enabled by the particle swarm optimization (PSO) technique, however, was found effective in alleviating local optima problem suffered typically by existing very fast block matching methods [45-52]. The PSO technique was introduced in [53, 54] as a robust stochastic optimization technique based on a social-psychological model of social influence and social learning [55, 56]. Belonging to the category of swarm intelligence methods, PSO is a population-based technique inspired by the social behavior and movement dynamics of flocks of birds, schools of fish, and herds of animals adapting to their environment. In PSO, a population of candidate solutions to the optimization problem, with their initial locations being randomly chosen in a search space, discovers optimal regions of the space through a process of

individuals' emulation of the successes of their neighbors. The available PSO-based motion estimation schemes found in the literature [45-52] suffer from several drawbacks. In video sequences, there is a high temporal correlation between the blocks of adjacent frames as well as a high spatial correlation between adjacent blocks of the same frame. Only spatial correlation is exploited in available PSO-based ME schemes [52] which use the found motion vectors of adjacent causal blocks for initializing the PSO particles of the current block. The PSO iterations, however, can achieve faster convergence if we exploit the temporal correlation with the collocated blocks in the adjacent frame as well. Moreover, within a video frame, motion is smooth and continuous which means that the estimated motion vectors between adjacent blocks are required to be correlated. In existing schemes, except for initialization, the PSO motion search is done separately for each block and no cooperation between adjacent blocks is allowed. Cooperation and communication between the blocks during the PSO process can ensure that the resulting estimated motion vectors of neighboring blocks are correlated. It can also speedup the convergence of PSO since the swarm of a given block can enhance its particles based on the knowledge received from adjacent blocks. In conclusion, there are plenty of strategies that can be incorporated into the dynamics of the PSO algorithm to improve its motion estimation accuracy and enhance its convergence speed.

1.4.2 Need for Distributed and Parallel BM Algorithms

Due to heavy computation demands of video coding, parallel implementation of the basic operations of this computation is necessary for satisfying the real time constraints usually imposed in multimedia applications. Moreover, the High Performance Computing (HPC) industry is marked by a relentless pursuit of ever greater levels of performance, driven by the never-ending

race toward scientific advancement. The march forward is often steady, as successive generations of technologies deliver incremental performance benefits over each previous version. HPC is now in the early stages of such a revolution. Single-core processors have given way to multi-core/many-core machine architectures, graphics processing units (GPU), and supercomputers. End users are still searching for the most effective ways to use them efficiently. This requires a change in the programming approach to develop ME algorithms with high parallelism in order to take advantage of the high speedup provided by the available hardware. In existing motion estimation algorithms, the use of previous macroblocks in the same frame for encoding the current macroblock makes ME an inherently sequential procedure, at the MB level, limiting the degree of parallelism that can be achieved. Effective techniques are needed to break the dependencies between the MBs in the frame without compromising the estimation quality. On the other hand, existing PSO-based BM algorithms [46-52] use centralized sequential processing within the MB. A central processor is needed to coordinate the particles of the swarm whose actions are updated in a sequential manner. Such a centralized approach hinders parallelism within the MB. Distributed PSO algorithms need to be explored for motion estimation that can achieve parallelism within the MB. In summary, novel BM algorithms are needed to achieve multi-level parallelism: parallelism within the MB as well as parallelism between the MBs in a frame.

1.4.3 Motion Estimation in High Resolution Video

Video resolution has witnessed a tremendous evolution. The majority of published evolutionary ME search algorithms only considers low resolution videos, as QCIF and CIF, in its experiments. However, the quality results of the ME algorithms can significantly change with the increasing of the video resolution. For low resolution videos, the quality results for ES and other

algorithms are very close. The great amount of pixels in high definition videos and the increase in the search area lead to an increase in the density of local minima on the error surface. This may lead BM algorithms to choose, more frequently, local minima as the best matching. Thus, the quality losses (in comparison with ES) are significant in this scenario. Techniques to avoid local minima falls in high resolution video must be explored to enhance the video quality without a significant increase in the ME computational complexity.

This thesis work follows a multidisciplinary approach by exploiting results from evolutionary optimization, game theory, diffusion adaptation in multi-agent networks, and parallel computing. The BM estimation problem is formulated as a non-convex optimization problem. Due to the non-convexity of the problem, evolutionary algorithms based on PSO are proposed to solve it. Game theory and diffusion adaptation are used to cast the problem in a distributed multi-agent framework and propose effective parallel algorithms to solve it. Concepts from the field of parallel computing provided parallelization strategies that are employed for developing the proposed parallel algorithms and their implementations.

1.5 Problem Definition

The main target of this thesis is to propose efficient ME algorithms with high accuracy and low computational complexity. PSO is an evolutionary algorithm that has shown promising results in the problem of block motion estimation. Efficient strategies are needed to enhance its performance. Moreover, the designed PSO-based ME algorithms should be inherently parallel. Several levels of parallelism need to be explored. Finally, novel algorithms with enhanced strategies need to be investigated to target HR video.

1.6 Thesis Contributions and Organization

In this section, we present the different thesis contributions and the associated chapters in the dissertation.

In Chapter 2, a cooperative motion estimation (ME) scheme using a modified Particle Swarm Optimization (PSO) algorithm is presented. The proposed algorithm is based on a multi-swarm PSO model where a swarm of PSO particles is defined for each macroblock (MB) in the frame. Motion Estimation is then performed in a cooperative manner concurrently for all the MBs in the frame. Cooperation between neighboring MBs during the motion estimation process is allowed through a communication step to exchange information about the motion vectors found so far in the estimation process. This synergic relationship between the swarms of adjacent MBs allows refining the motion search and leads to both a faster convergence of the PSO process and an improvement in the resulting motion vectors. Several techniques are also proposed to improve the search capacity and computational complexity of the PSO iterations. A novel PSO initialization scheme that exploits the existing temporal correlation is proposed to remove dependency between adjacent MBs. A fitness function history preservation mechanism is also presented to prevent redundant repeated calculations of the fitness function of a given search point by the PSO particles which dramatically decreases the computational complexity. The proposed scheme exhibits a high level of data parallelism since it is capable of performing motion estimation for all the MBs of the frame in parallel rather than serially. As a result, the presented algorithm is amenable to parallel processing techniques. In this chapter, a multicore implementation of the proposed algorithm is performed using the MATLAB® Parallel Computing Toolbox™ (PCT). Extensive simulations are performed to analyze the performance of the presented algorithm and its multicore implementation.

Chapter 3 introduces a novel parallel framework to speed up the BME process. This is

done by introducing a novel level of parallelism within the MB. The problem of BME is cast in a non-cooperative game-theoretic setting and a distributed multi-agent system is employed to solve the problem. First, a given MB is divided into subblocks and an agent is defined for each subblock. Then, the problem is formulated as a Consensus game and our approximation of the global utility function for the MB is defined. Building on this, agents' utilities are derived so that the resulting game is a potential game. To solve the game, distributed sequential and simultaneous algorithms based on game-theoretic Best Response Dynamics (BRD) and PSO are presented. Each agent uses PSO as its local search engine to autonomously maximize the utility of its subblock and BRD drive the agents with minimum local communication towards the maximum of the global utility function of the whole MB. Experimental results show that these algorithms provide good estimation quality with low computational cost as compared to other techniques. Moreover, in addition to its decentralized and distributed nature, the simultaneous algorithm is also inherently parallel at the agents' level within the MB. A thorough discussion and analysis of the proposed algorithms is included with a performance evaluation through extensive simulations. A parallel implementation of this algorithm using the MATLAB Parallel Computing Toolbox™ (PCT) on a multicore system is also provided to study the efficiency and speedup of the proposed parallel algorithm.

In Chapter 4, parallelism within the MB, which was solved in chapter 3 from a game-theoretic viewpoint, is tackled again but using concepts from diffusion adaptation in distributed multi-agent systems. We formulate and study the distributed optimization of block motion estimation using a network of cooperative nodes based on diffusion protocols. A modified diffusion-based PSO algorithm is proposed. Diffusion strategies are employed to allow the agents to cooperate and diffuse information in real-time in order to reach the common minimizer of the global cost function. A parallel implementation of this algorithm using the MATLAB PCT on a

multicore system is provided to study merits of the proposed scheme.

Chapter 5 targets the problem of BM estimation in HD video. It is first demonstrated that available PSO algorithms, when applied on High definition (HD) video, yield a quality worse than that obtained for low definition (LD) video. The reason behind this is that the problem of local minima becomes more significant as the resolution of the video increases and the existing ME schemes employ a basic version of PSO which is found to be not effective enough to combat the problem of local minima of HD video. In this chapter, we present a new ME scheme that employs a novel dynamic hybrid PSO algorithm. The PSO algorithm presented is hybrid in a sense that it employs improved strategies of the genetic algorithm (GA) like selection, mutation, and crossover to avoid being trapped in local minima. The algorithm is also dynamic since the maximum allowed velocity of the particles is dynamically varied in each iteration of the PSO process to effectively cover the search space. The presented algorithm is evaluated in terms of video quality and computational complexity and compared to existing fast searching ME techniques as well as existing PSO-based ME schemes.

In Chapter 6, we present the parallel implementation of the cooperative PSO algorithm, which was proposed in Chapter 2, on the NVIDIA GPU architecture using the CUDA platform. The NVIDIA programmable GPU has evolved into a highly parallel, multithreaded, many-core processor. Implementing the proposed cooperative PSO algorithm on the GPU is expected to yield a tremendous speedup.

Finally in Chapter 7, we summarize the contributions of this thesis work and outline some topics for future investigation.

This thesis includes 7 original papers that have been previously published/ submitted for publication in peer reviewed journals and conferences, as follows:

Table 1.1 List of thesis publications

Chapter 2	M. Jalloul and M. A. Al-Alaoui, "A Novel Cooperative Motion Estimation Algorithm Based on Particle Swarm Optimization and its Multicore Implementation ", Elsevier Journal of Signal Processing: Image Communication, vol. 39, part A, November 2015, pp.121-140.
	M. Jalloul, "A Parallel Computing Approach for Motion Estimation Based on Particle Swarm Optimization ", International Conference on Engineering of Reconfigurable Systems and Algorithms, ERSA-NVIDIA Award for Best Young Entrepreneur, ERSA 2013, Las Vegas, USA, July 22-15, 2013.
	M. Jalloul and M. A. Al-Alaoui, "A Novel Parallel Motion Estimation Algorithm Based on Particle Swarm Optimization", International Symposium on Signals and systems, ISSCS 2013, Romania, July 11-12, 2013.
Chapter 3	M. Jalloul and M. A. Al-Alaoui, "Agent-Based Game Theoretic Model for Block Motion Estimation and its Multicore Implementation," submitted to Elsevier Journal of Signal Processing: Image Communication, March, 2016.
	M. Jalloul and M. A. Al-Alaoui, "Block Motion Estimation and Potential Games ", International Conference on Image Processing, Computer Vision, & Pattern Recognition, IPCV 2015, part of WORLDCOM 2015, Las Vegas, USA, July 27-30, 2015.
Chapter 4	M. Jalloul and M. A. Al-Alaoui, "A Distributed Particle Swarm Optimization Algorithm for Block Motion Estimation Using the Strategies of Diffusion Adaptation", International Symposium on Signals and systems, ISSCS 2015, Romania, July 11-12, 2015
Chapter 5	M. Jalloul and M. A. Al-Alaoui, "A Novel Hybrid Dynamic Particle Swarm Optimization Algorithm for Motion Estimation in High Resolution Video ", International Conference on Engineering and Applied Sciences Optimization, OPT-i 2014, Kos, Greece, June 4-6, 2014.

CHAPTER 2

A NOVEL COOPERATIVE MOTION ESTIMATION ALGORITHM BASED ON PARTICLE SWARM OPTIMIZATION

In this chapter, a novel cooperative PSO algorithm is proposed for block motion estimation. The proposed scheme exploits spatial correlation by allowing the swarms of adjacent blocks to communicate during the PSO process and to exchange information about the motion vectors found so far. This collaboration allows for faster convergence and ensures that the resulting motion is smooth and continuous. Moreover, a novel initialization scheme is proposed that exploits temporal correlation by using motion vectors of collocated blocks in the previous frame. This method of initialization removes dependency between blocks of the same frame and makes the presented algorithm amenable to parallel processing methods. In addition, the adopted PSO iterations are designed to be dynamic by adaptively changing the maximum velocity, that limits the flying speed of the particles, which provides a balance between search exploration and exploitation. A fitness function history preservation technique is also proposed to prevent the redundant repeated calculations of the fitness function of a given search point by the PSO particles which provides a considerable reduction of the computational complexity.

The proposed algorithm, exhibits a high level of data parallelism and it is able to perform motion estimation for all the blocks of the frame in parallel. As a result, the proposed algorithm provides tremendous speedup if implemented on modern high performance computing (HPC) platforms ranging from multicore/many-core machine architectures to graphics processing units to supercomputers. In the literature, there have been several attempts to parallelize motion estimation [57-62]. Several works have proposed applying GPUs for motion estimation. Implementation of the ES motion estimation algorithm with OpenCL has been proposed in [57] and [58]. In [59],

implementations of the ES algorithm, the diamond search (DS) algorithm, and the four-step search (4SS) algorithms in CUDA have been proposed. A parallel implementation of the ES algorithm on the GPU using CUDA is also proposed in [60, 61] along with a parallel solution for multi-core processors using the Open Message Passing (OpenMP) library and a distributed solution for cluster/grid machines using the Message Passing Interface (MPI) library [61]. GPU-based hierarchical motion estimation in CUDA has been proposed in [62]. In this paper, we propose a parallel implementation of the proposed motion estimation scheme using the multicore capability of modern CPUs. A multicore implementation of the proposed scheme is implemented based on the Parallel Computing Toolbox of Matlab [63].The proposed parallel implementation is shown to be highly scalable and with more and more cores adopted in CPU, the algorithm speedup is expected to be higher. Moreover, the parallel performance of the proposed algorithm has been compared with that of the multicore implementation of the ES, 4SS, and DS algorithms which have also been implemented using Matlab PCT following the framework proposed in [61].

The rest of this chapter is organized as follows. Section 2.1 provides a brief review of the PSO algorithm. Section 2.2 presents the details of the proposed cooperative block motion estimation algorithm and section 2.3 provides the parallel implementation of the proposed algorithm using the Matlab environment. Section 2.4 shows the simulation results and presents an extensive evaluation of the performance of the presented algorithm. Finally, section 2.5 summarizes this chapter.

2.1 The General PSO Algorithm

The PSO technique was introduced in [53, 54] as a robust stochastic optimization technique based on a social-psychological model of social influence and social learning.

Belonging to the category of swarm intelligence methods, PSO is a population-based technique inspired by the social behavior and movement dynamics of flocks of birds, schools of fish, and herds of animals adapting to their environment. In the conventional PSO approach [53], the so-called swarm is composed of a set of particles that are placed in a search space where each particle represents a candidate solution to a certain problem or function. Initially, each particle is assigned a randomized velocity. The particles then ‘‘fly’’ through a multidimensional search space, where the position of each particle is adjusted according to its own experience and that of its neighbors. Each particle keeps track of its personal best location (p_{best}) in the problem space, which represents the best solution (fitness) it has achieved so far. The location of the overall global best value, obtained so far by any particle in the population, is called g_{best}. The PSO algorithm updates the position of a particle by moving the particle based on its past personal best (p_{best}) and the global best position (g_{best}) that has been found by all the particles in the swarm.

In an n-dimensional search space $S \subset \mathbb{R}^n$, and a swarm consisting of M particles, the i^{th} particle is in effect an n-dimensional vector

$$X_i = \{x_{i1}, x_{i2}, x_{i3}, \dots x_{in}\}^T \subset S. \quad (2.1)$$

The velocity of this particle is also an n-dimensional vector:

$$V_i = \{v_{i1}, v_{i2}, v_{i3}, \dots v_{in}\}^T \subset S. \quad (2.2)$$

The best personal position (p_{best}) encountered by the i^{th} particle is a point in S, denoted as

$$P_i = \{p_{i1}, p_{i2}, p_{i3}, \dots p_{in}\}^T \subset S. \quad (2.3)$$

In Particle Swarm Optimization with Inertia Weight Approach (PSO-IWA) [64], the velocity and position of a particle can be updated according to the following equations:

$$V_i(t+1) = wV_i(t) + c_1 r_1 [P_i(t) - X_i(t)] + c_2 r_2 \left[P_g(t) - X_i(t)\right], \quad (2.4)$$

$$X_i(t+1) = X_i(t) + V_i(t+1), \quad (2.5)$$

where i is the index of the particle, i = 1, 2. . . M; w is the inertia weight which balances the local and global search during the optimization process. It is linearly decreasing with iterative generations as in:

$$w = w_{max} - (w_{max} - w_{min}) \times \frac{t}{N}, \quad (2.6)$$

where t is the current iteration and N is a predefined maximum number of iterations. The maximal and minimal weights w_{max} and w_{min} are usually set to 0.9 and 0.4; c_1, c_2 the positive acceleration constants; r_1, r_2 the random numbers, uniformly distributed within the interval [0, 1]; g the index of the best positioned particle among the entire swarm; P_i the position of p_{best} for the particle i; and P_g is the position of g_{best} for the entire swarm. A maximal flying speed v_{max} is used to restrict the flying of the particles.

2.2 Proposed Cooperative Motion Estimation Algorithm Using PSO

In this research work, we propose a new block matching algorithm based on a novel cooperative PSO approach. A multi-swarm model is presented where a swarm of PSO particles is allocated for each MB in the frame. A modified PSO algorithm is applied to all MBs concurrently for a certain number of iterations. After that, cooperation between swarms of adjacent MBs is allowed through a synchronization step which is performed among neighboring MBs to exchange information about the motion vectors (MVs) found so far in the PSO process. Some of the PSO particles are re-initialized according to the received information. Based on the assumption that the motion field is smooth and varies slowly, there are strong correlations between motion vectors of the neighboring blocks. As a result, this synchronization step allows making use of the spatial correlation characteristic between neighboring MBs to refine the MVs found so far in the PSO process. A second stage of PSO iterations is then performed concurrently for all the MBs. The

whole process ends whenever termination conditions are reached. The steps of the proposed scheme are explained below.

2.2.1 Particle Initialization

A swarm consisting of M particles is generated for each MB. Each particle of a given MB represents a matching MB within the search window in the reference frame. Using the PSO iterations, the positions of the particles is continuously updated until the global minimum of the mean square error (MSE) cost function is reached. In the standard PSO algorithm, the initial population is randomly selected, which brings high computational complexity to the motion search since the iterations are starting from random points which might be far from the global minimum. However, if the initial points are chosen to be close to the optimum, then faster convergence can be achieved. Li [65] and Xiao [66] demonstrated that the use of solutions generated through some domain knowledge to set the initial population (i.e. non-random solutions) can significantly improve its performance. In [52], authors proposed an initialization scheme that exploits the existing spatial correlation between neighboring MBs where the particles of a given MB are initialized using the estimated motion vectors of its adjacent neighboring MBs. This mode of initialization imposes a dependency constraint between the MBs of the same frame and thus hinders parallelism. In this paper, a novel initialization scheme is proposed to remove any dependency between the MBs. Since motion vectors have a high temporal correlation feature, we initialize 9 particles of each MB to the MVs of the collocated MB in the previous frame as well as its 8 adjacent neighbors. We also initialize one of the particles to the (0, 0) MV to account for static blocks. The rest of the M particles are randomly generated. Therefore, for an MB at location (i,j) in frame t, we initialize the positions of its M particles as follows:

$$\{x_1, x_2, x_3, \dots, x_{10}\} =$$

$$\left\{Mv_{i-1,j-1}^{t-1}, Mv_{i-1,j}^{t-1}, Mv_{i-1,j+1}^{t-1}, Mv_{i,j-1}^{t-1}, Mv_{i,j}^{t-1}, Mv_{i,j+1}^{t-1}, Mv_{i+1,j-1}^{t-1}, Mv_{i+1,j}^{t-1}, Mv_{i+1,j+1}^{t-1}, (0,0)\right\} \quad (2.7)$$

$\{x_{11}, \dots, x_M\}$= random position within the search area. (2.8)

Notice that at this point, we cannot use the MVs of the adjacent blocks in the same frame since these MVs are not calculated yet and the only apriori information we have is the motion of the MBs of the previous frame.

It should also be noted that since this information is still not available for the second frame in the video sequence, then our implementation applies the proposed algorithm starting from the third frame. Motion estimation for the second frame is performed using ES to obtain accurate motion vectors to be used for the initialization step of frame number three in the proposed algorithm. This initialization step is shown in Fig. 2.1.

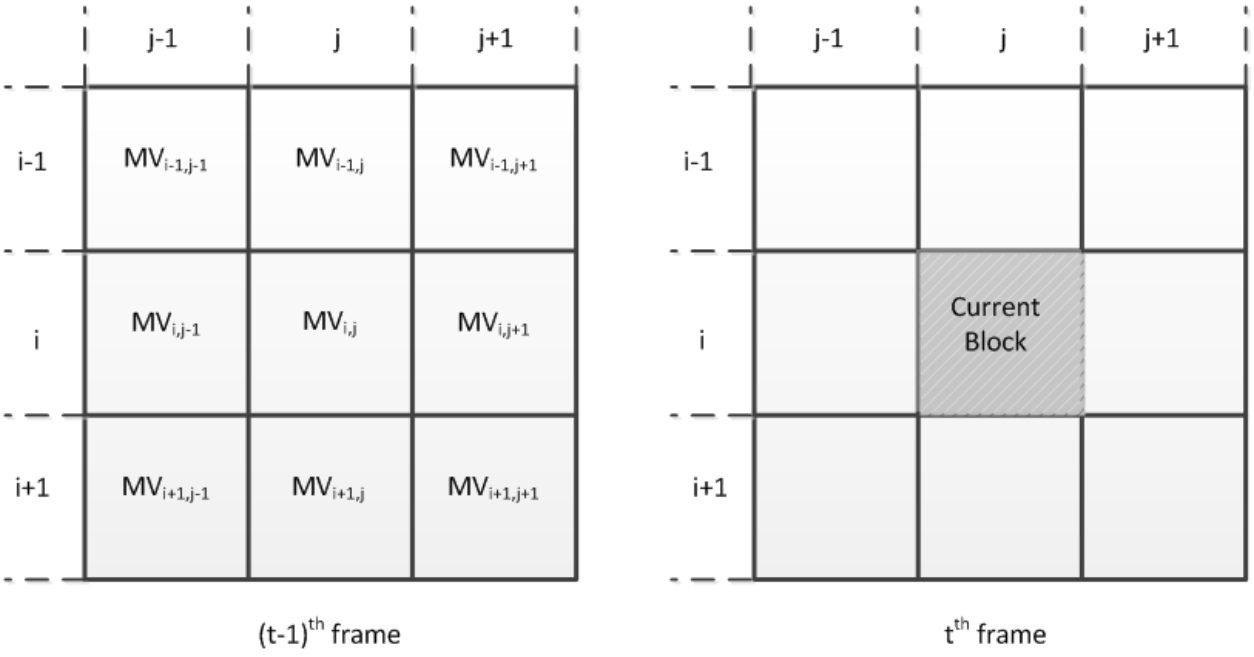


Figure 2.1 Initialization of the positions of the particles of the current MB in frame t using the motion vectors of collocated MB and its eight neighboring MBs in frame (t-1).

2.2.2 First Stage of a Modified PSO Process

After initialization, the swarms of particles of all MBs go through a modified PSO process where they are allowed to run for a predefined N_t number of iterations in parallel. During each iteration, each MB with index j adjusts the positions and velocities of its particles according to (2.4) and (2.5), independently from other MBs, evaluates the fitness function at the new positions, then it updates the values of Pij and Pgj which are the positions of the best fitness attained so far for particle i and the global best position for MBj respectively.

2.2.2.1 Adaptively-Varied Maximum Velocity

The maximum velocity which limits the flying speed of the particles is adaptively changed in this modified version of PSO. A dynamic control of the maximum velocity v_{max}, described in Section 2.1, can provide a balance between search exploitation and exploration. In the PSO process, a large v_{max} allows to better explore the complete solution space; on the contrary, a small v_{max} directs the method to perform a local search. Therefore, in this modified PSO algorithm, a higher v_{max} value is adopted in the early stage of the search process and a lower value later to perform a local search. A linearly decreasing function is adopted to gradually reduce the v_{max} value in the current iteration in proportion to the iteration number, this is given by:

$$v_{max}(t) = \frac{V_{max}}{t}, \qquad (2.9)$$

where V_{max} is an empirically determined value and t is the iteration number.

2.2.2.2 Fitness Function History Preservation

In our algorithm, fitness function history preservation is proposed to avoid unnecessary redundant fitness function calculations of search points that have been visited before by any

particle during the PSO process. This is done as follows. A two-dimensional array of dimensions$(2 \times p + 1)^2 \times 2$ is defined, where p is the search parameter which defines the search area. The rows of the array represent all the possible search points in the defined search area of a given MB. Values in the first column are binary, either zero or one, to show if the corresponding search point has been visited before. Values in the second column of the array are the values of the fitness function calculated for the corresponding search points incase these points were visited before. During the PSO process, before the fitness function of a certain position is evaluated, the array is checked to see if that position was visited before. In that case, re-evaluation of the fitness function is skipped and the fitness value saved in the array is used. Otherwise, the value of the fitness function of that position is evaluated and the corresponding entries of the array are updated. In this way, the fitness function of a certain position within the search area is evaluated only once. Usually the trajectory of a particle during the PSO process can cross a search position more than once. Moreover, other particles can reach that position as well. By following the proposed scheme, the number of fitness function evaluations during the PSO process is dramatically decreased which plays a key role in reducing the computational complexity of the whole algorithm. In [52], a similar process, called particle history preservation, was used. In that process, for each particle in the PSO process, a binary array of the same size as the search area is used to keep track of the positions that have been visited before by that particle. During the iterative process, for each particle, its corresponding array is checked to see if the position reached was visited before. In that case, re-evaluation of the fitness function is avoided and that search position is skipped. In [52], the fitness values are not recorded. In our proposed algorithm however, only one array is used for all the particles, and the fitness values of each search point visited by any particle is recorded and subsequently used whenever that search point is reached. In this way, fitness function evaluation

of a given point is skipped because of repeated crossing of that position by, not necessarily the same, but any particle during the PSO process.

2.2.2.3 Termination Conditions

The first stage of the proposed PSO process terminates whenever the maximum number of iterations N_t is reached. Early termination of search is allowed whenever the fitness value of the global best position is less than a predefined threshold value T_{th} and when the fitness value associated with the Pgj position remains the same for K_{max} iterations, even if the maximum iteration number N_t is not yet reached.

The procedure for implementing the first stage of the proposed PSO process can be summarized in pseudo code as shown in table 2.1.

Table 2.1 Pseudo code of the first stage of the proposed PSO process

For each frame **do**
For each block **do**
Initialize the fitness history array entries to zeros
Initialize particle velocities to zeros
Initialize particle positions as shown in Fig. 1 and (2.7)-(2.8)
Repeat
For each particle i=1,…,M **do**
Check its flag in the history array
If the flag is 0 then
Calculate fitness function
Update P_i and P_g
Save the value of the fitness value in the history array
Set flag to 1
Else
Retrieve the value of the fitness function from the history array
Update P_i
End if
Adaptively change v_{max} using (2.9)
Update the velocity using (2.4)
Update the position using (2.5)
End for
Until stopping conditions are met
End for
End for

2.2.3 Cooperation between Neighboring Swarms

After the first stage of PSO iterations is completed by all MBs of the frame, a cooperation step is performed to coordinate the estimation process of neighboring swarms. This is done by exploiting the high spatial correlation existing between MVs of neighboring blocks. To do that, each MB's swarm sorts its M particles in a decreasing order according to their fitness values. Then the last 8 particles which have the highest (worst) fitness values are eliminated and replaced by 8 new particles which are initialized to the global best positions, Pg, values of its 8 neighboring swarms. In this synchronization step, each swarm is allowed to refine its motion search process using information from neighboring swarms. Weak particles having the worst fitness values are replaced with strong particles which are located closer to the global optimum. This cooperation is expected to speed up the convergence of the PSO algorithm since the learning process is now supervised and guided by the information received from the neighboring blocks. Communication between neighboring MBs is required in this step where each MB will broadcast to its 8 neighbors the value of its global best location, Pg, found so far in the motion search process. This process is shown in Fig. 2.2.

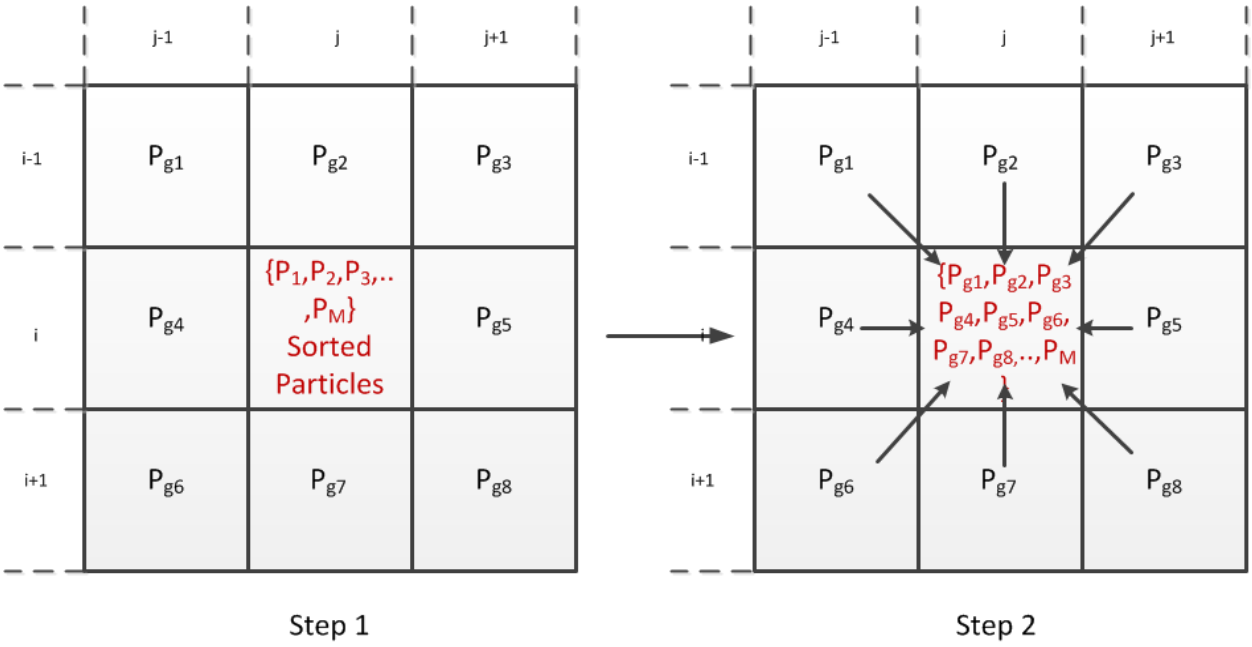


Figure 2.2 Cooperation with neighboring Swarms. In step 1, particles are sorted according to their fitness values. In step 2, the first 8 worst particles are replaced with new ones with positions initialized with the P_g values of neighboring MBs.

2.2.4 Second Stage of a Modified PSO Process

After synchronization, the updated swarm of particles of each MB is allowed to go through another stage of the modified PSO process. This stage combines the information received from the neighboring MBs during synchronization with the information gained from the first PSO stage to find a better optimal solution. The same termination conditions as those used in the first stage of PSO are applied here.

2.3 Parallel Implementation of the Proposed Algorithm

The proposed cooperative framework for block motion estimation is iterative and self-organized where each PSO swarm for each MB is autonomous. Dependency between MBs is limited to the cooperation phase where synchronization between neighboring swarms is performed through a communication step. This makes the proposed scheme amenable to parallel processing

methods. In this section, a multicore implementation of our proposed algorithm is proposed using the MATLAB® Parallel Computing Toolbox™ (PCT). MATLAB PCT [63] can solve computationally and data-intensive problems using multicore processors, GPUs and computer clusters. It provides high level constructs such as parallel for-loops, special array types and parallelized numerical algorithms to parallelize MATLAB applications without CUDA or MPI programming. The flowchart of the proposed parallel implementation is shown in Fig. 2.3.

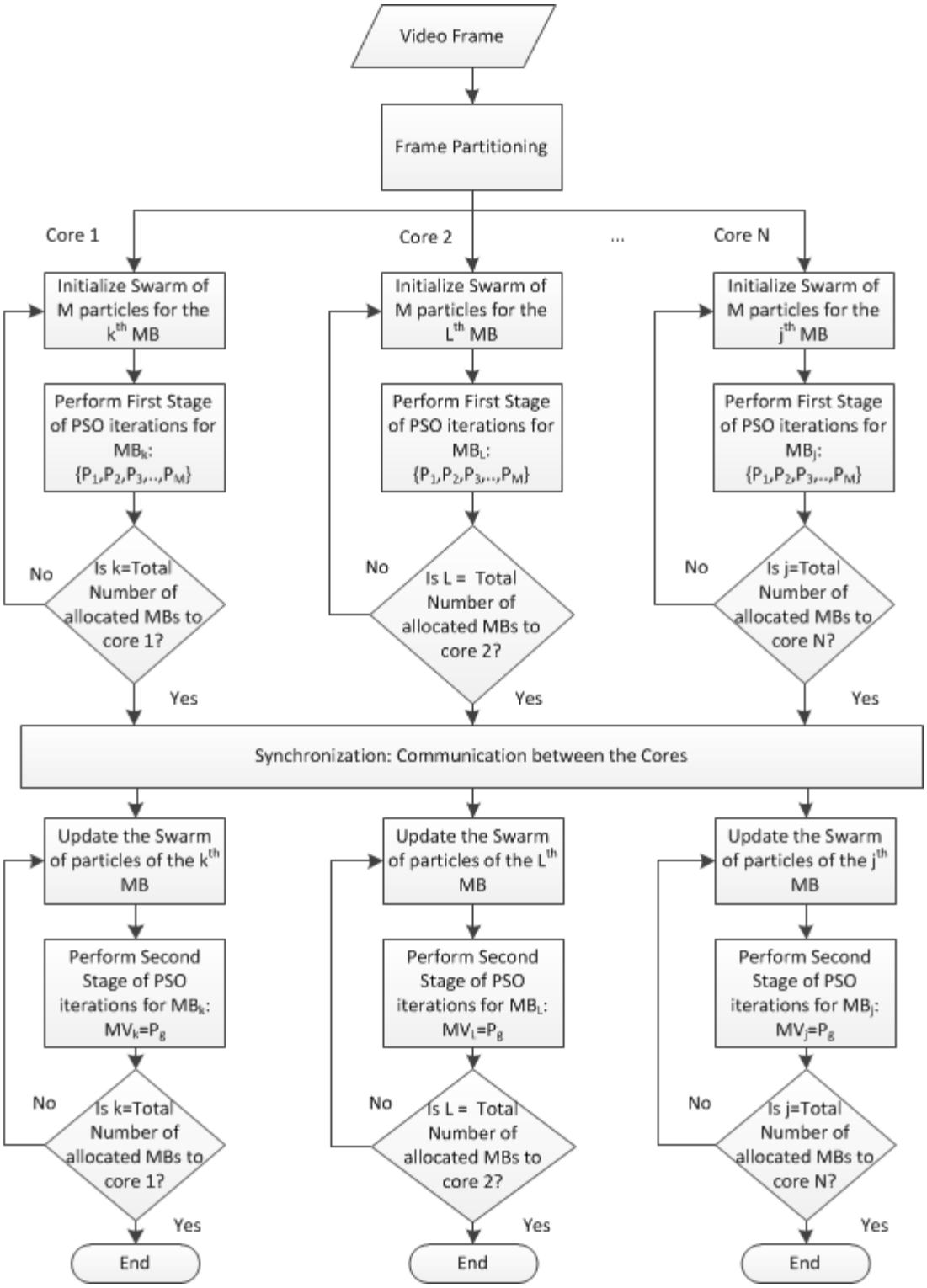


Figure 2.3 Flow Chart of the proposed parallel implementation.

2.3.1 Frame Partitioning Using Co-Distributed Arrays

One of the first steps in designing a parallel algorithm is to break the problem into discrete "chunks" of work that can be distributed to multiple tasks. This is known as decomposition or partitioning. There are two basic ways to partition computational work among parallel tasks: domain decomposition and functional decomposition. In our proposed parallel algorithm, domain decomposition is chosen. In this type of partitioning, the data associated with a problem is decomposed. Each parallel task then works on a portion of the data simultaneously. In our implementation, a given frame is divided into 16x16 macroblocks (MB). These MBs are evenly partitioned among the available computing resources or CPU cores to ensure load balancing. There exist different possible domain-partitioning schemes. The chosen scheme has to account for two fundamental issues: load balance, and communication balance [67]. In our algorithm, a block partitioning along the rows of the frame is chosen where the rows of the frame are evenly partitioned among the available cores. The reason why this partitioning scheme is chosen is to minimize the communication overhead required between the cores. In our algorithm, as explained before, each MB needs to communicate with its direct neighbors for synchronization. If these neighbors were assigned to the same core as the central MB, then the communication overhead is reduced. Block partitioning along the columns would also yield the same benefits. In MATLAB, frame partitioning is performed using *co-distributed arrays.* A *co-distributed array* is an array partitioned into segments, with each segment residing in the workspace of a different lab. An even partitioning of the MBs of the frame among the MATLAB labs ensures load balancing which decreases possible idle times. Partitioning along the rows of a given frame among the cores is shown in Fig. 2.4.

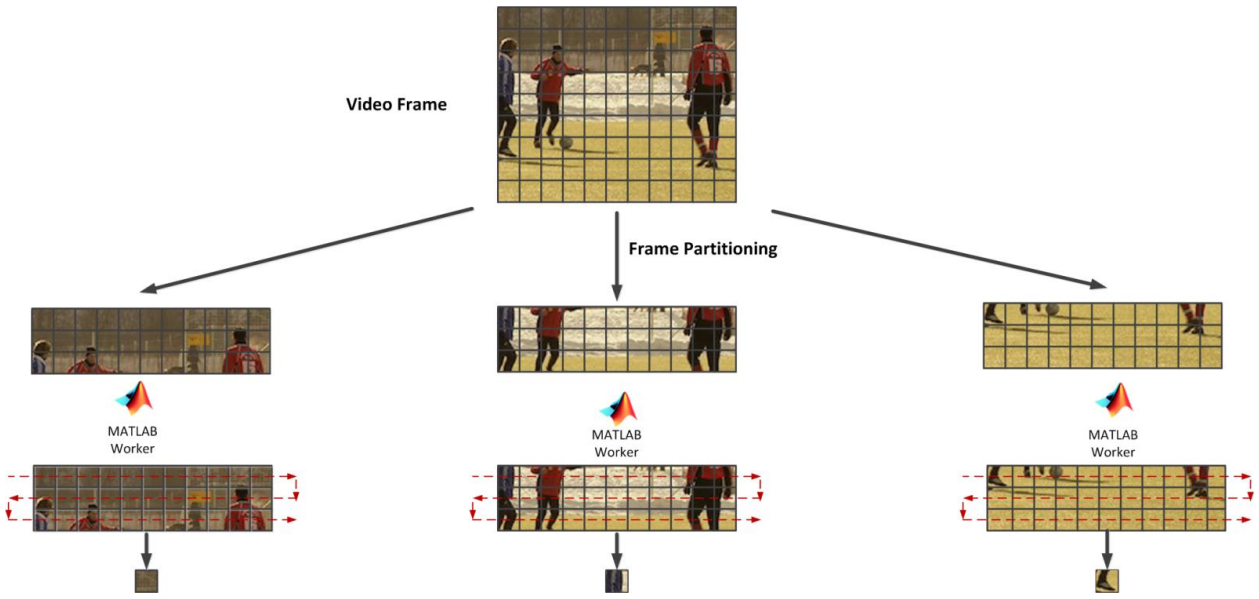


Figure 2.4 Frame partitioning among the processing cores.

2.3.2 Parallel MB Processing Using Looping Over a Distributed Range (for-drange)

When a for-loop over a distributed range is executed in a parallel job, each lab performs its portion of the loop, so that the labs are all working simultaneously. Because of this, no communication is allowed between the workers while executing a for-drange loop. In particular, a lab has access only to its partition of a codistributed array [63]. In our implementation, the frame is partitioned equally along the rows of MBs among the different *labs*. Therefore, we have used the *for-drange* construct to allow the simultaneous processing of each lab of its assigned rows of MBs in the frame.

2.3.3 SPMD Block

The Matlab toolbox provides the single program multiple data (*spmd)* construct and several message-passing routines based on an MPI standard library (MPICH2). The *spmd* construct allows designating sections of the code to run concurrently across workers participating

in a parallel computation. During program execution, *spmd* automatically transfers data and code used within its body to the workers and, once the execution is complete, brings results back to the MATLAB client session [63]. This allows us to implement data-parallelism where each Matlab *lab* executes the same lines of code but on different sections of the video frame. If the number of available cores is high enough, then each Matlab *lab* would operate on a different MB of the frame and we would be able to perform motion estimation for all the MBs in parallel. Within the *spmd* block, communication or synchronization is allowed between labs.

2.3.4 Communication and Cooperation between the Labs Using labSendReceive

During the cooperation step of our algorithm, each MB is required to broadcast to its 8 neighboring MBs the global best position, P_g, found so far in the optimization process. As a result, each MB will also receive from each of its 8 neighbors the value of the global best position acquired by that neighbor. This interlab communication within the parallel job is implemented using *labSendReceive*. The environment query functions *labindex* and *numlabs* here are equivalent to*MPI_Comm_rank* and *MPI_Comm_size*. Since in our algorithm each lab operates on a certain number of rows of MBs, then each lab needs to communicate only with the previous lab to send to it the values corresponding to its upper row of MBs and receive from it the values corresponding to the lower row of MBs in that lab. Similarly, it should communicate with the next lab to send to it the values corresponding to its lower row of MBs and receive from it the values corresponding to the next lab's upper row of MBs. Because the migration topology used in our method is a stepping stone model, the adjacent labs need only to send and receive data from each other in a cyclic pattern. The function *labSendReceive* proposes a perfect solution for this pattern, which is designed to enable the cyclic type communication, or any paired exchange, to be written more

simply. Moreover, the deadlocking behavior is also prevented effectively [63].

2.4 Simulation Results

2.4.1 Simulation Setup

Several test video sequences of various formats and various motion intensity, (QCIF: 176x144), LD (CIF: 352x288), SD (480p: 832x480), and HD (720p: 1280x720) downloaded from [68, 69], have been used to test the performance of our proposed algorithm and compare it to existing techniques. Results are presented with two distinct criteria: execution time and objective motion estimation quality. Quality is more important in high end solutions such as video broadcasting whereas for low cost solutions, execution time (or algorithm complexity) must be kept low. Snapshots of the used test video sequences are shown in Fig. 2.5 and their properties are given in Table 2.2.

Soccer (176x144 and 352x288)

Bus (176x144 and 352x288)

Tennis (352x288)

Stefan (352x288)

Foreman (352x288)

Container (352x288)

RaceHorses (832x480)

Parkrun (1280x720)

Figure 2.5 Test video sequences.

Table 2.2 Test Sequences Used in the Simulations

Test Video Sequence	Format	Frame Size	Frame Rate	Motion type
Soccer	QCIF	144x176	15	Fast
Bus	QCIF	144x176	15	Very Fast
Soccer	CIF	144x176	30	Fast
Bus	CIF	288x352	30	Very Fast
Tennis	CIF	288x352	30	Very Fast
Stefan	CIF	288x352	30	Very Fast
Foreman	CIF	288x352	30	Medium
Container	CIF	288x352	30	Slow
RaceHorses	SD 480p	832x480	30	Very Fast
Parkrun	HD 720p	1280x720	50	Medium

The proposed algorithm is simulated on a server with two Intel® Xeon® E5520 2.66GHz CPUs (total of 8 cores) and 32GB RAM. The execution platform is Matlab R2012a.

In our simulations, every frame is divided into MBs of size 16 * 16 pixels. The search step-size is one integer pixel and we used one reference frame which is the previous frame. The search parameter p which defines the search area is chosen to be 15 for all the tested sequences except for the HD (Parkrun) video sequence where p was chosen to be 31. The reason behind this choice is that picking a small value of p for the HD video sequence would yield very poor results for motion estimation since the search area is very small as compared to the resolution and it might not contain the global optimum. The algorithm can be easily extended to use arbitrarily sized blocks, smaller step-sizes and multiple reference frames.

2.4.2 PSO Parameters

For the PSO algorithm, the size of the particle population was chosen to be M=10 initialized according to the scheme described in Section 2.2.1.

For N_t and K_{max}, as discussed in Section 2.2.2.3, using a very small N_t (and K_{max}) can lead

to very fast convergence of the PSO-based algorithm, albeit at the expense of low block matching accuracy. On the other hand, using a large N_t (and K_{max}) could improve the accuracy at the cost of increased computational cost. Moreover, varying N_t and K_{max} also affects the speedup of the proposed algorithm since using a small N_t (and K_{max}) means that a lower amount of computation work is being done as compared to the amount of communication work which is independent of the values of N_t (and K_{max}). This would reduce the overall speedup obtained from a multicore system and would diminish the scalability of the algorithm. By balancing these conflicting requirements, the PSO-based method could provide a good overall performance over a wide range of video sequences, when the maximum iteration number was chosen to be N_t = 3, and K_{max} = 2.

The pre-set minimum MSE error, T_{th} (mentioned in Section 2.2.2.3), is another empirically determined threshold that can regulate the accuracy/complexity tradeoffs. If the threshold is too large, the algorithm tends to run fast at the cost of a lower accuracy. In our simulations, the threshold for MSE, T_{th}, was chosen to be 7.

The maximum allowed velocity for the PSO particles, v_{max}, is dynamically varied in every iteration of the proposed modified PSO process, as mentioned in Section 2.2.2.1. It is initially set to V_{max} and then linearly decreased according to the iteration number. If V_{max} is too high, particles might fly past good solutions; if it is too small, particles may not explore sufficiently beyond local solutions. In the literature, v_{max} was fixed to about 10–20% of the dynamic range of the variable on each dimension [65]. In our simulations, for a search range of ±15, we chose V_{max} = 15 as a starting value for the maximum allowed velocity which adaptively decreases in each iteration. The average value of the maximum velocity allowed during the entire process ($2 \times N_t$ iterations) is 6.125 which is around 19.76% of the dynamic range. For a search range of ±31, we chose V_{max} = 31 as a starting value which gives an average value of 12.65 during the entire process

($2 \times N_t$iterations) which is around 20% of the dynamic range.

2.4.3 Motion Estimation Quality

Objective motion estimation quality is measured interms of Peak Signal to Noise Ratio (PSNR) values averaged over the first 100 frames of each test video sequence. Such value indicates the reconstruction quality when motion vectors, which are computed through a BM approach, are used. In PSNR, the signal comes from original data frames whereas the noise is the error introduced by the calculated motion vectors. The PSNR is thus defined as:

$$PSNR = 10 \times log_{10}\left(\frac{255^2}{MSE}\right), \quad (2.10)$$

where *MSE* is the mean squared error between the original frames and those compensated by the motion vectors.

Table 2.3 gives the average PSNR results for the ES algorithm and several traditional fast searching techniques, like TSS [16], 4SS [18], DS [19], and ARPS [22]. PSNR results are also given for the recently proposed PSO-based ME algorithms given in [51, 52]. The simulation results presented are based on the averages of the data (PSNR and search point) obtained from 50 repeated runs of the PSO-based algorithm to strengthen the statistical significance. Increasing the number of runs also yield only very negligible changes to the averages which do not differ significantly. Simulation results show that the proposed algorithm provides an improvement in motion estimation quality as compared to the other techniques. Fig. 2.6, Fig.2.7, Fig. 2.8, Fig.2.9, Fig. 2.10, Fig. 2.11, Fig. 2.12, and Fig. 2.13 show that the proposed algorithm can closely follow the PSNR values of the ES method on the frame-by-frame basis.

Table 2.3 Motion estimation quality in terms of PSNR of the proposed approach as compared to existing techniques

Sequence	ES	TSS	4SS	DS	ARPS	PSO [52]	PBPSO[51]	Proposed
Soccer, QCIF, 15fps, p=15	25.014	24.02	22.106	23.267	23.761	24.33	20.12	24.55
Bus, QCIF, 15 fps, p=15	23.3505	21.8863	19.7652	20.4179	21.0361	22.8094	17.579	23.079
Soccer, CIF, 30fps, p=15	30.1983	28.2174	27.0174	27.6907	28.6623	29. 3297	21.847	29. 574
Bus, CIF, 30 fps, p=15	25.608	22.373	19.789	20.337	21.793	24.925	18.448	25.403
Tennis, CIF,30 fps, p=15	29.198	26.859	27.764	28.128	28.070	28.224	24.304	28.6003
Stefan, CIF, 30fps, p=15	26.9370	24.6205	23.702	23.981	26.0389	26.482	20.238	26.518
Foreman, CIF, 30fps, p=15	34.6824	33.4948	33.809	34.243	34.185	34.174	31.257	34.329
Container, CIF, 30fps, p=15	32.8433	26.8720	23.5904	23.547	29.113	32.386	18.573	32.701
RaceHorses, 480p, 30 fps, p=15	29.338	26.809	24.891	26.018	27.445	28.86	21.425	29.018
Parkrun, 720p, 30 fps, p=31	25.613	20.436	23.661	23.314	25.33	24.4	19.094	25.487

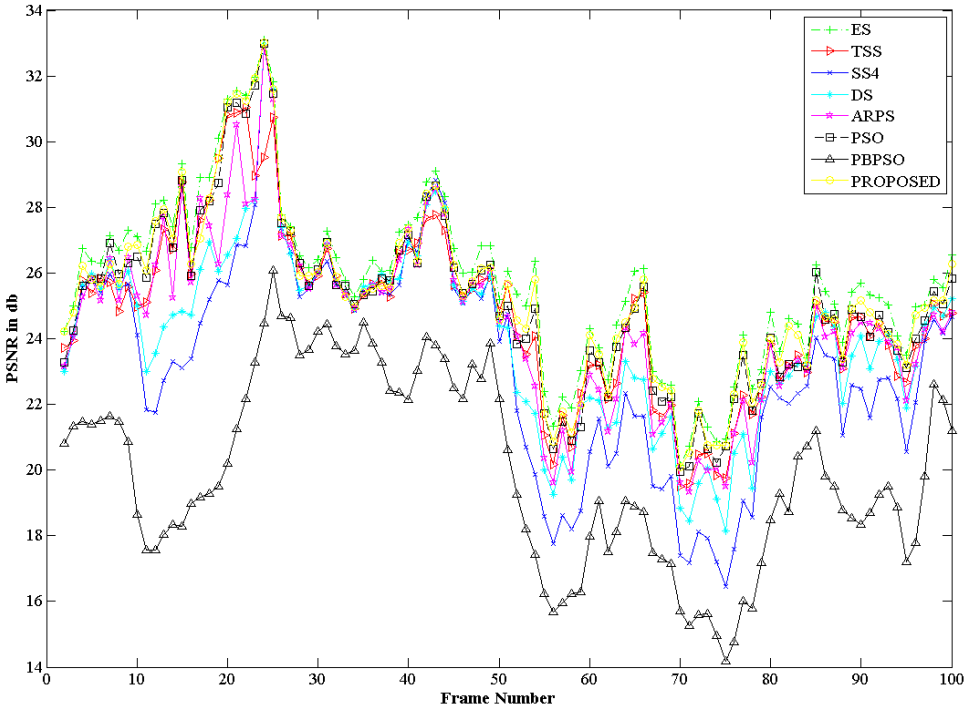


Figure 2.6 Motion estimation accuracy measured in PSNR for “Soccer, QCIF” sequence.

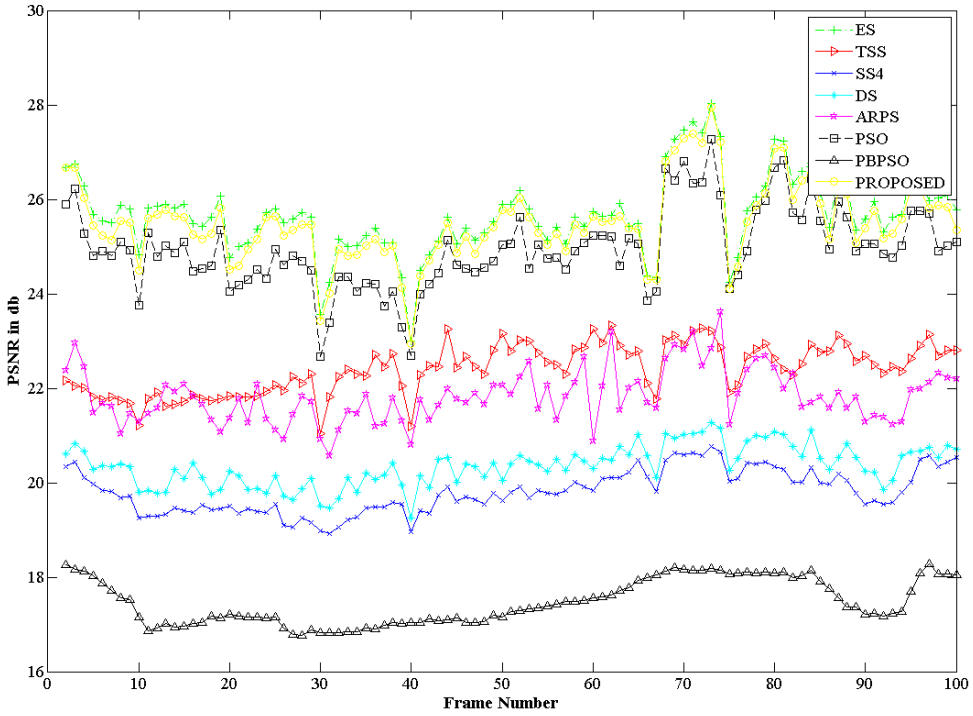


Figure 2.7 Motion estimation accuracy interms of PSNR for “Bus, CIF” sequence.

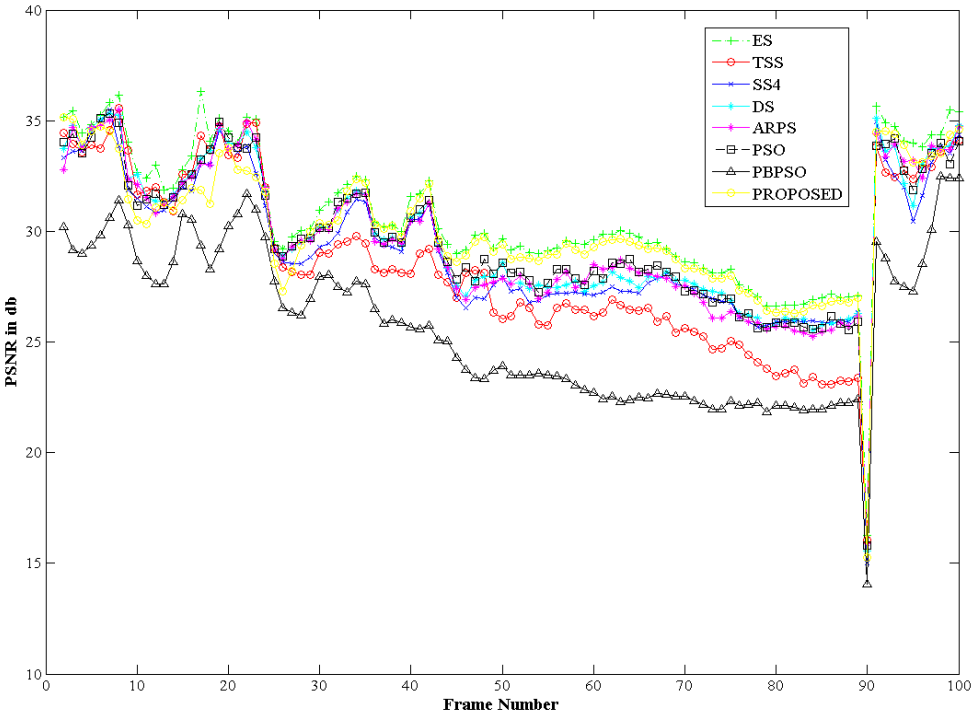


Figure 2.8 Motion estimation accuracy in terms of PSNR for “Tennis, CIF” sequence.

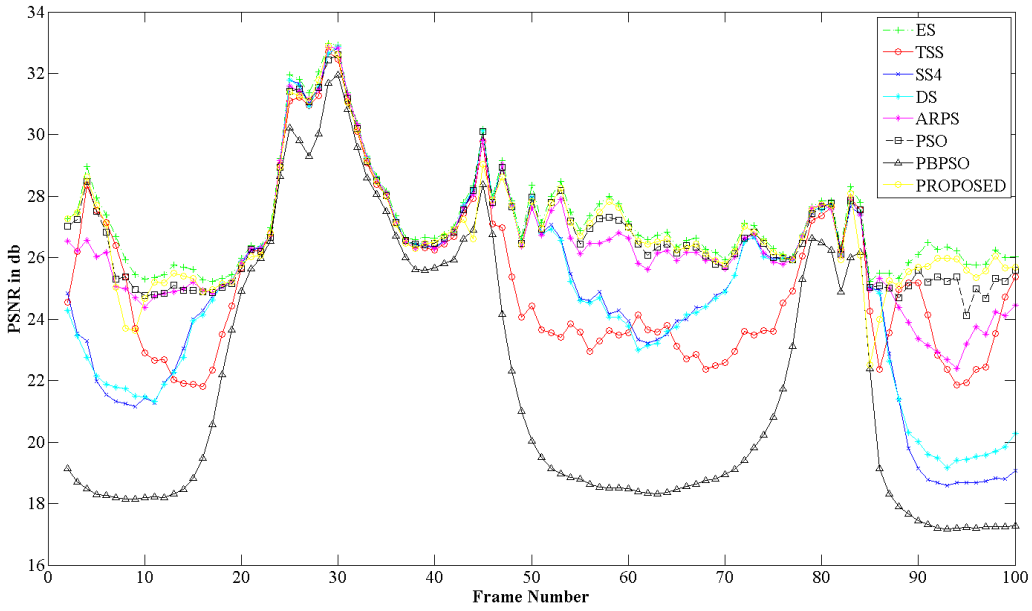


Figure 2.9 Motion estimation accuracy interms of PSNR for “Stefan, CIF” sequence.

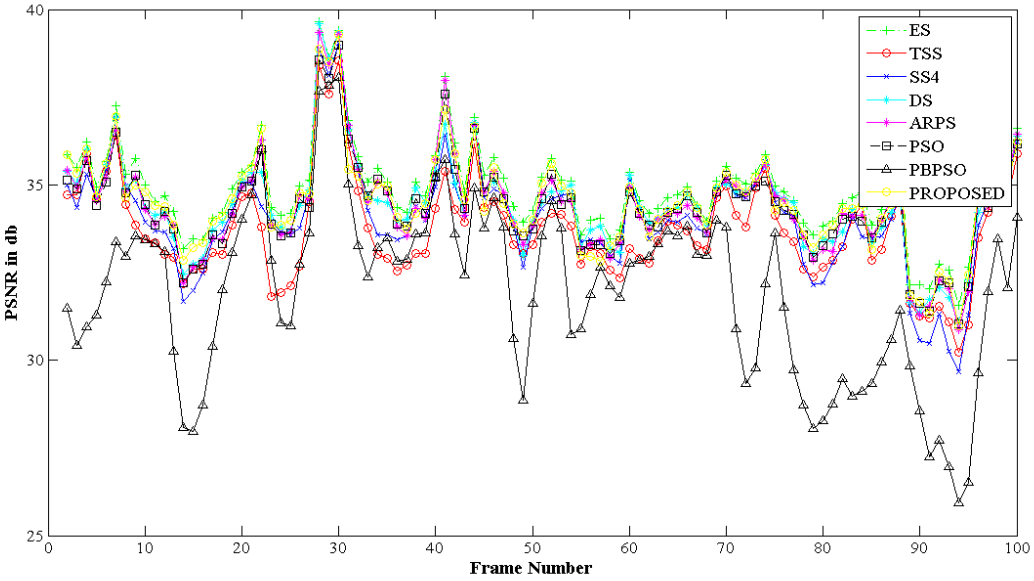


Figure 2.10 Motion estimation accuracy interms of PSNR for “Foreman, CIF” sequence.

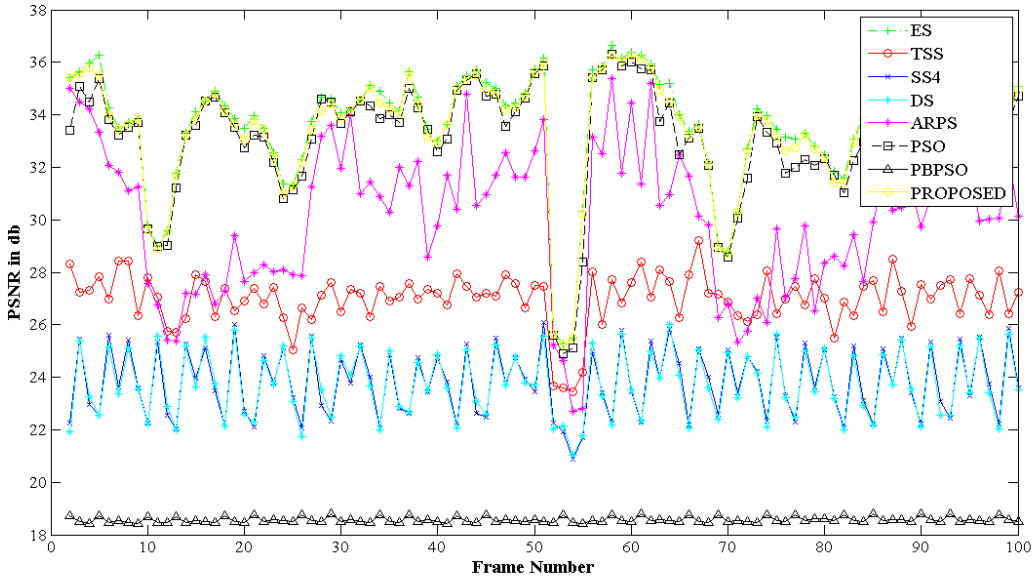


Figure 2.11 Motion estimation accuracy interms of PSNR for "Container, CIF" sequence.

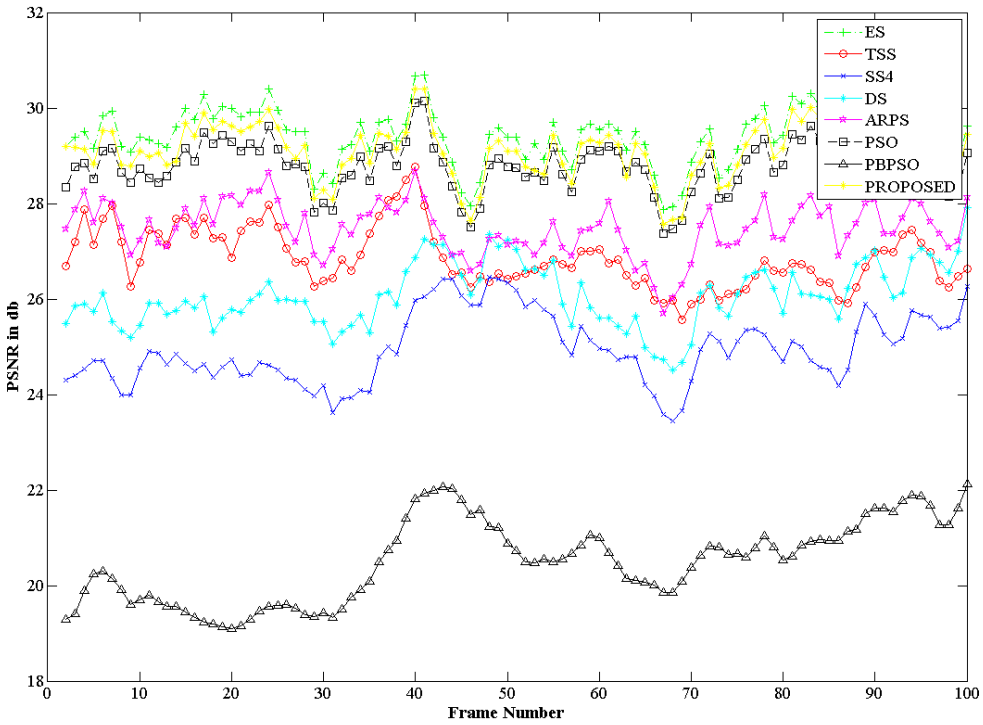


Figure 2.12 Motion estimation accuracy interms of PSNR for "RaceHorses" sequence.

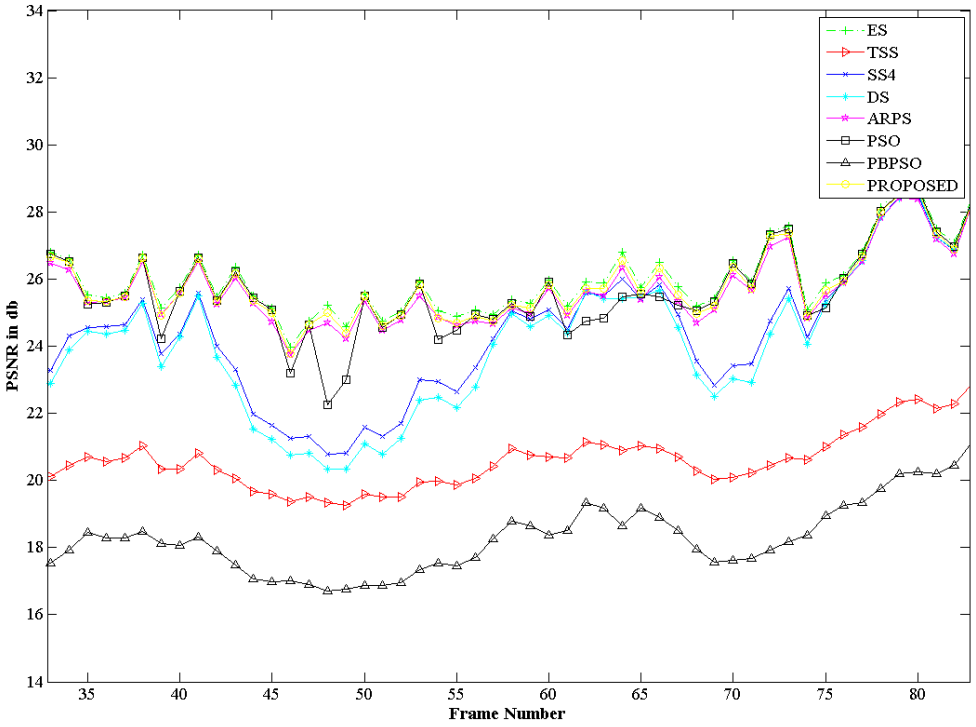


Figure 2.13 Motion estimation accuracy interms of PSNR for "Parkrun" sequence.

In order to show the good performance of the proposed algorithm more intuitively, the reconstructed images of the fifth frame of Racehorses sequence are shown in Fig. 2.14. It can be seen from Fig. 2.14, that the fifth frame images of Racehorses restructured by TSS, 4SS, DS, ARPS, as well as PBPSO [51] and PSO[52], all miss some details, and the images restructured by the proposed algorithm have retained the details of the original image which was translated into the highest PSNR value. Note that the frame reconstructed by PBPSO [51] is almost the same as the previous frame (reference frame). The reason behind this is that PBPSO [51] initializes the PSO particles within the small zone around the center block assuming that most blocks are stationary or semi-stationary. Consequently, the early termination strategies adopted in PBPSO lead to an early convergence to positions around the center. For sequences with fast motion, like RaceHorses, PBPSO would produce low estimation quality. The performance accuracy of PBPSO is also

expected to deteriorate with the increase in video resolution where the assumption that motion vectors are within a small zone around the center is no longer valid.

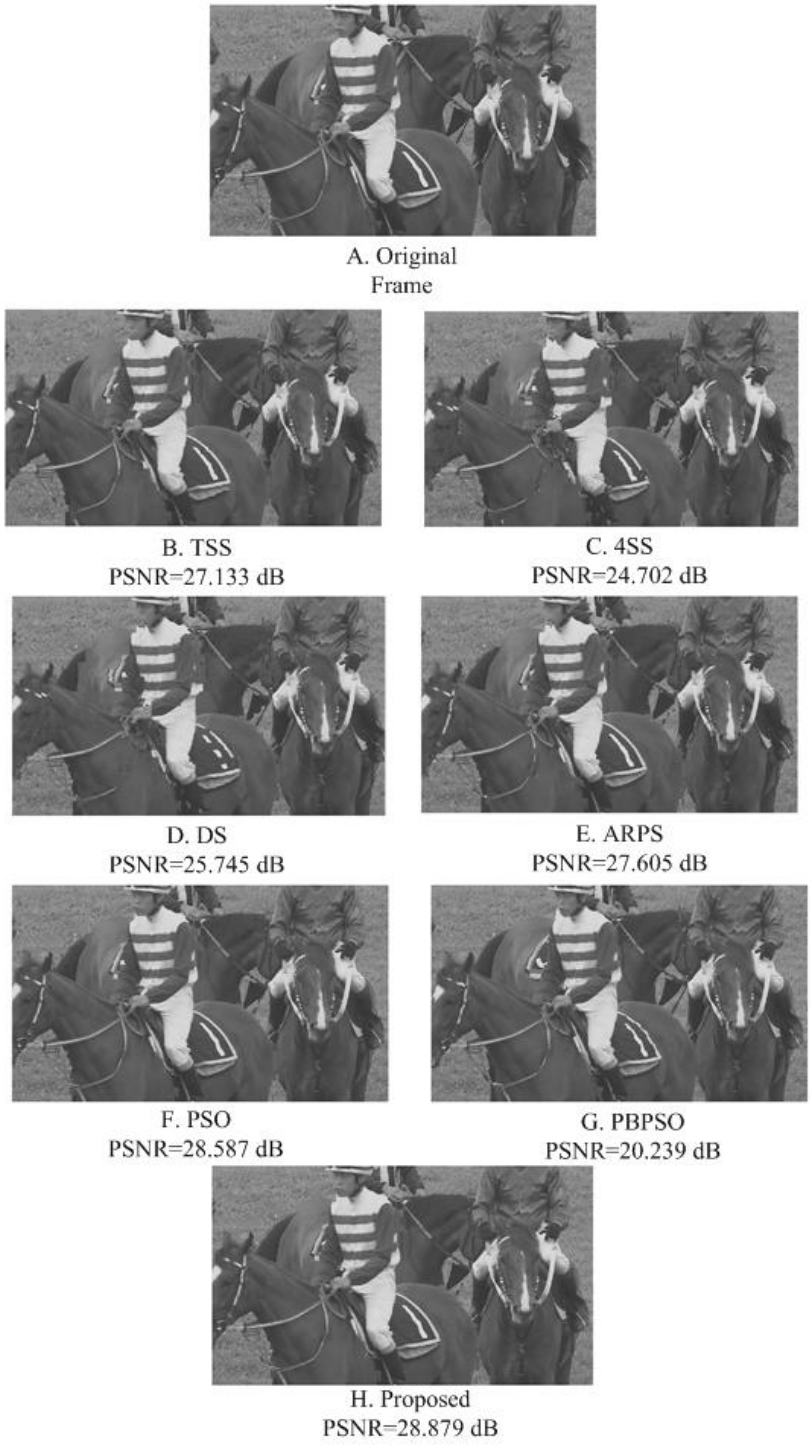


Figure 2.14 The reconstructed images of the fifth frame of RaceHorses by using different algorithms.

2.4.4 Computational Complexity

In block matching motion estimation, the average number of candidate blocks checked for each MB is used as the evaluation criterion of computation complexity. In this paper, the average number of fitness function evaluations for each MB is used as a metric of the computational complexity. The simulation results of a single core implementation of our algorithm are compared with the results of existing algorithms and the results are listed in Table 2.4.

It can be seen in Table 2.4 that the ES method searches every candidate block within the search window, so it needs to search 961 points for each MB if p=15 and 3969 points if p=31. TSS, 4SS, and DS are based on fixed template so the number of search points is relatively less. On average, they need to search 20–40 points. The PSO-based ME algorithm proposed in [52] gives less number of search points than the fast search methods and PBPSO given in [51] provides further reduction in the computational complexity. In our proposed algorithm, the exploitation of time-space correlation of video sequences through effective particle initialization and synchronization, fitness calculation history preservation, and the efficient termination strategies used have decreased the number of search points needed dramatically. As shown in Table 2.4, it ranges between 7 and 11 for the different video sequences. Theoretically, for $2 \times N_t = 6$, and M=10, the maximum number of fitness function evaluations is 60, but as shown in Table 2.4, the needed points are much less because of the effective strategies adopted in the proposed algorithm.

Table 2.4 Comparison of the average number of fitness function evaluations per block for various algorithms based on the first 100 frames of the video sequences

Sequence	ES	TSS	4SS	DS	ARPS	PSO [52]	PBPSO[51]	Proposed
Soccer, QCIF,15fps, p=15	961	29.33	18.33	17.66	13.25	16.2	11.024	9.47
Bus, QCIF,15fps, p=15	961	29.46	19.85	22.51	12.44	19.22	12.482	11.17
Soccer, CIF,30fps, p=15	961	31.13	20.01	19.89	10.61	13.54	12.24	6.998
Bus, CIF, 30fps, p=15	961	31.23	24.24	21.39	12.35	17.83	11.92	7.64
Tennis, CIF,30 fps, p=15	961	30.973	18.893	17.195	9.448	15.839	12.243	9.156
Stefan, CIF, 30fps, p=15	961	30.753	18.803	18.023	8.819	14.098	11.289	7.457
Foreman, CIF, 30fps, p=15	961	30.7602	18.456	16.661	8.978	12.925	12.114	9.368
Container, CIF, 30fps, p=15	961	31.165	21.247	23.0984	9.7977	12.337	12.751	6.333
RaceHorses, 480p, p=15	961	32.17	30.06	23.35	14.97	16.6	13.31	9.88
Parkrun, 720p, p=31	3969	40.11	22.97	21.18	9.77	15.924	12.182	6.86

2.4.5 Parallel Performance

The average execution times of the proposed algorithm per frame using Matlab PCT are shown in Table 2.5. The algorithm is simulated using different Matlab workers, or labs, and simulation times are recorded. T_0 is the overhead time needed to setup the parallel environment and to create *codistributed* arrays. T_1 is the time needed to perform the first stage of PSO iterations. T_2 is the time needed for synchronization or communication. Finally, T_3 is the time needed to perform the second stage of PSO iterations. The parallel performance of our algorithm is evaluated interms of the speedup factor, parallel efficiency, percentage of time savings, and granularity.

Table 2.5 Parallel performance of the proposed algorithm using Matlab PCT

Sequence	Number of Labs	T0 (s)	T1 (s)	T2 (s)	T3 (s)	Total Time (s)	Speedup	Efficiency %	Granularity
Soccer QCIF, 15 fps, p=15									
	1	0.0153	0.0875	0.004	0.0783	0.1698	1	100	-
	3	0.2905	0.0354	0.0127	0.0337	0.0818	2.075	69.193	5.440
	9	0.369	0.0264	0.02	0.0278	0.0742	2.288	28.605	2.71
Bus, CIF, 30 fps, p=15									
	1	0.0155	0.3083	0.0039	0.2716	0.5838	1	100	-
	2	0.2342	0.1667	0.0129	0.1517	0.3313	1.762	88.107	24.682
	3	0.288	0.1129	0.0135	0.0998	0.2262	2.580	86.030	15.755
	6	0.3176	0.0667	0.0164	0.0624	0.1455	4.012	66.872	7.871
	9	0.3761	0.0541	0.025	0.0412	0.1203	4.852	60.660	3.812
RaceHorses, 480p, 30 fps, p=15									
	1	0.0157	1.2485	0.0037	1.0851	2.3373	1	100	-
	2	0.288	0.6688	0.015	0.5897	1.2735	1.835	91.766	83.9
	3	0.293	0.4495	0.0181	0.3973	0.8649	2.702	90.079	46.784
	6	0.3418	0.2732	0.0198	0.2314	0.5244	4.457	74.285	25.484
	10	0.387	0.2288	0.0209	0.1804	0.4301	5.434	67.929	19.578
Parkrun, 720p, 50 fps, p=31									
	1	0.0163	2.4847	0.004	3.321	5.8097	1	100	-
	3	0.2905	0.9012	0.014	1.1621	2.0773	2.796	93.225	147.378
	5	0.336	0.584	0.0212	0.7984	1.4036	4.139	82.782	65.207
	9	0.368	0.462	0.027	0.5144	1.0034	5.790	72.375	36.162

2.4.5.1 Speedup

To measure the parallel performance of our proposed algorithm, we used the speedup factor, *S(n),*which is defined as:

$$S\ (n) = \frac{T_s}{T_n}, \tag{2.11}$$

where T_s is the total execution time on a single processor, while T_n is the total execution time on a multicore system of n processors. In Table 2.5, T_s is taken as the total time when executing the code on one *lab* and T_n is the total time when executing the code on *n labs*. Fig. 2.15 shows a plot of the speedup as function of the number of cores for the four sequences. An interesting

observation in the plot is that the rate of increase of the speedup grows with the increase of video resolution. Moreover, for the same number of cores, a higher speedup is achieved for higher resolutions. The reason behind this is that, for the same number of cores, increasing the resolution would increase the amount of computational work done by each core since more MBs would be assigned to the cores, while the communication overhead remains approximately the same. This means that the parallelizable portion of the algorithm increases with the increase of the video resolution. According to Gustafson's law [70], limitations imposed by the sequential part of a program may be countered by increasing the total amount of computation. It states that if α, the sequential fraction of the parallel execution time, is small, the speedup is approximately equal to the number of processing cores, as desired. It may even be the case that α diminishes as the number of cores, *n*, (together with the problem size) increases; if that holds true, then *S* approaches *n* monotonously with the growth of *n*. In our proposed algorithm, as the video resolution, or problem size, increases, the computational fraction of the algorithm increases with the increase in the video resolution and thus α decreases. Therefore, the proposed algorithm allows an increase in the maximum theoretical speedup achieved as the video resolution increases.

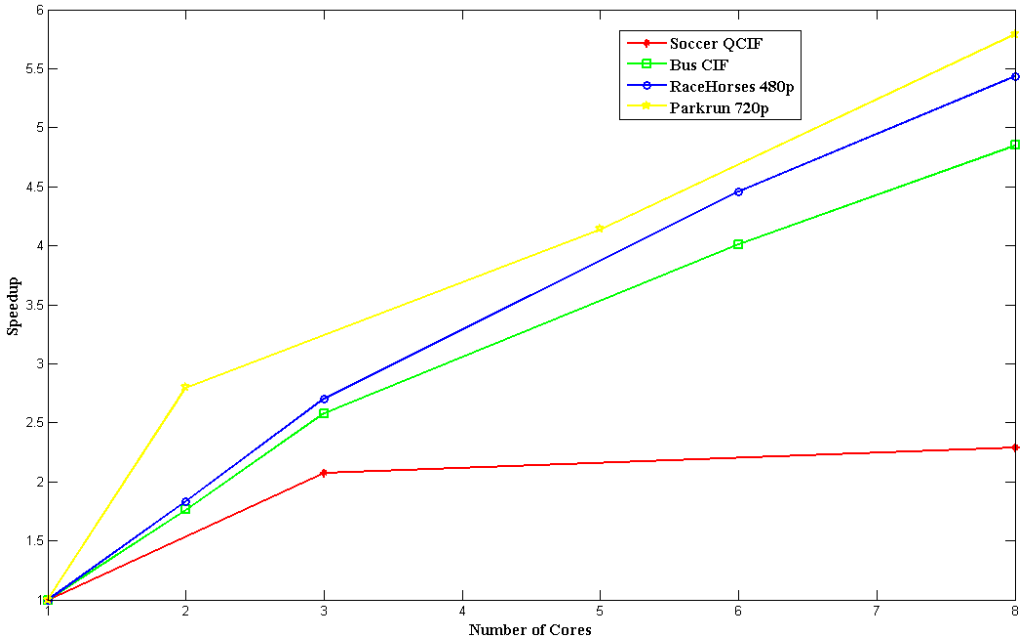


Figure 2.15 Speedup achieved by the proposed parallel implementation on different number of cores.

2.4.5.2 Parallel Efficiency

The parallel efficiency, E(n), describes the fraction of the time that is being used by the processors for a given computation. It is defined as:

$$E(n) = \frac{S(n)}{n} * 100. \quad (2.12)$$

It should be noted that when calculating the efficiency for *9* and *10 labs*, *n* is taken as 8 since only 8 physical cores are available. It is observed that the parallel efficiency decreases with the increase of the number of cores. This is due to the fact that as the number of cores increases, the probability that each of the 8 neighboring MBs of a given block would be allocated to a different core increases. As a result, this leads to an increase in the needed inter-processor communication during the cooperation stage of the algorithm. On the other hand, it is found that the parallel efficiency increases with the increase of video resolution since more MBs would be allocated to

each core.

2.4.5.3 Granularity

In parallel computing, granularity, *G*, is a qualitative measure of the ratio of computation to communication, which is given by:

$$G = \frac{T_{Computation}}{T_{Communication}} \ . \quad (2.13)$$

Granularity is said to be *coarse* if relatively large amounts of computational work are done between communication events, and it is said to be *fine* if relatively small amounts of computational work are done between communication events. To measure the granularity of our algorithm, the computation and communication times are taken as:

$$T_{Computation} = T_1 + T_3, \quad (2.14)$$

$$T_{Communication} = T_2. \quad (2.15)$$

As shown in Table 2.5, the granularity increases with the increase of the resolution of the video frame since more data will be assigned to the cores which results in an increase in the computation time. On the other hand, it decreases with the increase of number of cores since less number of MBs would be assigned to each core which results in a decrease in the computation time. Moreover, as the number of cores increases, the amount of communication needed also increases which results in an increase in the communication time. The communication time however has an upper limit since each core is required to communicate to a maximum of 8 other cores during the synchronization phase. It is worth mentioning here that the communication cost of the coordination step is considered low since it is limited to the exchange of the P_g values of between an MB and its 8 direct neighbors only. Overall, in the presented implementation granularity is considered coarse which implies more opportunity for performance increase.

2.4.5.4 Theoretical Analysis of Parallel Performance

The parallel performance of the proposed implementation depends on the computation complexity of the PSO estimation process and the communication overhead of the cooperation stage. The computational complexity of this ME approach depends on the number of fitness function evaluations performed. This is directly related to the population size,M, and the maximum number of total iteration,N_t, allowed. Theoretically, we have a maximum of $2 * M * N_t$cost function evaluations required for each MB. Nevertheless, because this approach exploits the spatial and temporal correlations of motion vectors, refines the motion search process through inter-swarm cooperation, and allows an early termination condition for the MBs, it is estimated that the PSO algorithm will converge before N_t is reached. We assume that the time taken for a swarm of a given MB to converge is Twhich is proportional to $2 * M * N_t$. On the other hand, the communication cost of the inter-swarm cooperation stage is due to the exchange of the values of $P_g$between an MB and its 8 direct neighbors only. The amount of communication overhead incurred in this method is $\mathrm{K} * \mathrm{D}$ where D is the network time to communicate a value of P_g between two cores and K is the number of neighboring MBs that are allocated to a different core. In general, the value ofKdepends on the number of MBs allocated per core in the frame. For a frame with $W * H$ number of MBs and assuming that all MBs need $2 * M * N_t$ iterations to converge, then the speedup of this algorithm for n cores can be estimated to be:

$$S_{proposed\ algorithm} = \frac{T*W*H}{T*(\frac{W*H}{n})+(K*D)}. \quad (2.16)$$

If the number of available cores is large, i.e. $n \geq W * H$, as in the case of many-core systems like GPUs, then we will have one MB allocated per core, $\frac{W*H}{n} = 1$, and the value of K is 8 in this case. The theoretical speedup will be:

$$S_{proposed\ algorithm,many-core\ system} = \frac{T*W*H}{T+(8*D)} \quad (2.17)$$

It is noticed that the speedup in this case is proportional to the resolution of the video. This leads to a very important conclusion. As the resolution of the video increases, the use of many-core machines allows for a linear increase in the speedup. For example, for the Ultra High Definition TV (UHDTV) or 4k resolution (3840*2160), the value of $W * H$ is 32400 MBs of size 16*16. The parallel implementation of our proposed algorithm on the massively parallel architecture of modern GPUs, which consists of thousands of efficient cores, is expected to yield a tremendous improvement in performance for today's UHDTV resolution.

2.4.5.5 Comparison with Existing Parallel ME Algorithms

The computational complexity of the proposed algorithm is also compared with existing parallel ME algorithms. Multicore versions of the ES, 4SS, and DS algorithms have been implemented using Matlab PCT following the framework proposed in [61]. MB level parallelism is exploited where the MBs in the frame are evenly partitioned among the available processing cores. Note that the parallel ES algorithms proposed in [57-61], and implemented using OpenCL [57,58] and CUDA[59-61], are also based on MB level parallelism along with search point parallel processing to compute the cost of each search point. Simulation results are given in terms of the average number of fitness function evaluations per lab for a given frame based on the first 100 frames of every sequence. These results are shown in Table 2.6. It is shown that the proposed algorithm achieves a dramatic reduction in the computational costs per lab for the different video sequences under different parallel simulation scenarios.

Table 2.6 Average number of fitness function evaluations per lab for a given frame based on the first 100 frames of each sequence

Sequence	Number of Labs	Parallel ES	Parallel 4SS	Parallel DS	Proposed
Soccer, QCIF, p=15					
	1	77439	1673	1575	920
	3	25813	557	525	309
	9	8605	186	175	106
Bus, CIF, p=15					
	1	344256	9914	8692	3448
	3	114752	3322	2903	1142
	9	38251	1107	968	378
RaceHorses, 480p, p=15					
	1	1423800	46829	36380	16280
	3	474600	15610	12127	5467
	10	142380	4683	3638	1629
Parkrun, 720p, p=31					
	1	13572364	78394	66059	51917
	3	4524121	26131	22020	17323
	9	13572000	9.35E+03	10770	7193

2.5 Summary

In this chapter, a novel cooperative block motion estimation algorithm based on PSO is presented. The proposed scheme exploits spatial correlation by allowing the swarms of adjacent blocks to communicate during the PSO process and to exchange information about the motion vectors found so far. This collaboration allows for faster convergence and ensures that the resulting motion is smooth and continuous across neighboring blocks which translates into a better estimation quality. Moreover, a novel initialization scheme is proposed that exploits temporal correlation by using motion vectors of collocated blocks in the previous frame. This method of initialization removes dependency between blocks of the same frame which makes the presented algorithm amenable to parallel processing methods. In addition, the adopted PSO iterations are designed to be dynamic by adaptively changing the maximum velocity, that limits the flying speed of the particles, which provides a balance between search exploration and exploitation. A fitness

function history preservation technique is also proposed to prevent the redundant repeated calculations of the fitness function of a given search point by the PSO particles which provides a considerable reduction of the computational complexity. The performance of the proposed algorithm is found to be superior, in terms of both accuracy and computational complexity, as compared to existing fast searching algorithms and state of the art PSO-based motion estimation schemes. Moreover, we presented an efficient and highly scalable implementation of the proposed cooperative motion estimation algorithm using the Matlab PCT environment on a local shared-memory multicore system. Results of simulations showed that a speedup of 6.33 can be achieved for HD sequences using 8 cores. The proposed algorithm is shown to be highly scalable. It also allows an increase in the maximum theoretical speedup achieved as the video resolution increases. The multicore performance of the proposed scheme is also compared with existing parallel algorithms in the literature and is shown to give superior results.

CHAPTER 3

AGENT-BASED GAME THEORETIC MODEL FOR BLOCK MOTION ESTIMATION AND ITS MULTICORE IMPLEMENTATION

This chapter introduces a novel parallel framework to speed up the BME process. This is done by introducing a novel level of parallelism within the MB. A given MB is divided into subblocks and an agent is defined for each subblock. The main target of this chapter is to show how a system of autonomous agents can, in a distributed fashion and relying only on local interactions, optimize the global objective function of the whole MB.

The last years have witnessed an intense research activity in the development of novel distributed algorithms for multi-agent systems with performance guarantees. A particular effort has been devoted to the study of game-theoretic approaches that can model and regulate selfish agent interactions [71]. By means of these, the multi-agent coordination objective is formulated in terms of Nash Equilibria (NE), which corresponds to the natural emergent behavior arising from the interaction of selfish players. Due to their modularity, game dynamics can easily be implemented by agents relying on local information, leading to a robust performance. In particular, we tackle this problem by first defining a global utility function for the whole MB. Each agent has its own private utility function of its subblock which it aims to maximize. However, these are defined such that, for any unilateral switch in strategy, an agent's change in payoff is equal to the change in the global utility. Consequently, the global maximum is a Nash equilibrium (i.e. it is a stable solution to the game). In this way, selfish agents can be used to solve an inherently cooperative problem, because their self–interest drives them towards solutions with higher global utility. Furthermore, we derive the agents' utilities from the approximate global utility function

such that the agents play a *potential game* [72] at each time-step. The potential game concept provides a valuable theoretical framework for distributed multi-agent cooperation problems. First, a game that admits the potential property is guaranteed to possess a Nash equilibrium. Second, from the definition of a potential game, the Nash equilibrium for every local cost function is consistent with the global objective. The potential game framework, therefore, provides distributed optimization problems with theoretical support for problem simplification [73]. A best-response approach is a common method in potential games to achieve a Nash Equilibrium Point (NEP) [74]–[75]. The idea of best-response dynamics (BRD) is that every player produces its best response in terms of the current state of the other players [76]. The proposed best-response dynamics enable players to make autonomous decisions to optimize their local utility functions by monitoring the actions of their neighbors. In order to carry out this optimization step, agents should be equipped with local optimization capabilities. Due to the non-convexity of the agent's utility function, we resort to modern optimization techniques. PSO is chosen as the global optimization algorithm used by the agents due to its profound intelligence and simple algorithmic structure. Each agent is equipped with a PSO engine that finds its best response at each time step given the actions of the other players by optimizing its local utility function. Two distributed algorithms are proposed in this chapter based on two versions of BRD: the sequential and simultaneous BRD. The main distinction between these two algorithms lies in the mode and frequency of the inter-agent communication needed during the process to update neighborhood information which also determines the level of dependency between the agents. The proposed sequential algorithm possesses convergence guarantees to the NE of the underlying potential game, whereas the proposed simultaneous algorithm contains a high level of data parallelism at the agent level which makes it highly amenable to parallel implementations. A multicore parallel implementation of the

proposed simultaneous motion estimation scheme is presented using MATLAB® Parallel Computing Toolbox™ (PCT) [63].

The chapter is organized as follows. In Section 3.1, we provide a general background on game theory. Section 3.2 then presents the proposed game–theoretic framework. In this section, we formulate the problem as a Consensus game, and describe our approximation of the global utility function. Building on this, we show how to derive agents' utilities so that the resulting game is a potential game, and describe the learning algorithms that can be used to solve it. In section 3.3, two distributed algorithms based on simultaneous and sequential BRD are proposed to solve the game-theoretic formulation of BME, and their convergence properties are analyzed. Then, in Section 3.4, a multicore implementation of the proposed simultaneous algorithm is presented using the MATLAB® PCT. Section 3.5 shows the simulation results and presents an extensive evaluation of the performance of the presented algorithms. Finally, Section 3.6 summarizes this chapter.

3.1 Game Theory

Game theory is a branch of mathematics aimed at the modeling and understanding of the interactions between several decision-makers (called players) who can have conflicting or common objectives. A game is a situation in which the benefit or cost achieved by each player from an interactive situation depends, not only on its own decisions, but also on those taken by the other players [77]. Essentially, the theory splits into two branches: non-cooperative and cooperative game theory. The distinction between the two is whether or not the players in the game can make joint decisions regarding the choice of strategy. Non-cooperative game theory is closely connected to minimax optimization and typically results in the study of various equilibria, most

notably the Nash equilibrium. Cooperative game theory examines how strictly rational (selfish) actors can benefit from voluntary cooperation by reaching bargaining agreements. Another distinction is between static and dynamic game theory, where the latter can be viewed as a combination of game theory and optimal control. In general, the theory provides a structured approach to many important problems arising in signal processing and communications, notably resource allocation and robust transceiver optimization. Recent applications also occur in other emerging fields, such as cognitive radio, spectrum sharing, and in multihop-sensor and adhoc networks [78-79].

3.2 The Proposed Game Theoretic Framework

The problem addressed in this chapter is how to solve the optimization problem of BME in a distributed way through a network of autonomous players in a game theoretic framework.

3.2.1 The Definition of the Game

3.2.1.1 Global Objective Function:

$$J_{glob}(\hat{u},\hat{v}) = \sum_{j=0}^{N-1}\sum_{i=0}^{N-1}|g_t(x+i,y+j) - g_{t-1}(x+\hat{u}+i,y+\hat{v}+j)|, \qquad (3.1)$$

where $g_t(.)$ is the gray value of a pixel in the current frame I^t and$g_{t-1}(.)$ is the gray level of a pixel in the previous frame I^{t-1}.That is our global objective is to find one motion vector for the whole block (of dimension NxN). To do that using game theory, we propose to decompose the block into K subblocks and then associate each subblock to a player. Each player would be trying to find the motion vector for its subblock. We want the players at the end of the game to reach consensus that is they should all agree on a common motion vector which is the minimizer of the global objective function. The game should allow the players to communicate during the search

process.

3.2.1.2 Agents

Consider a networked multi-agent system where agents are labeled by $k \in P = (1,2,\ldots\ldots,K)$. Each agent is associated to a subblock. The cost function of agent k is the Sum of Absolute Difference of the subblock of dimension LxL at position (x_k, y_k)as shown in Fig. 3.1:

$$J_k(\hat{u},\hat{v}) = \sum_{j=0}^{L-1}\sum_{i=0}^{L-1}|g_t(x_k+i, y_k+j) - g_{t-1}(x_k+\hat{u}+i, y_k+\hat{v}+j)| \qquad (3.2)$$

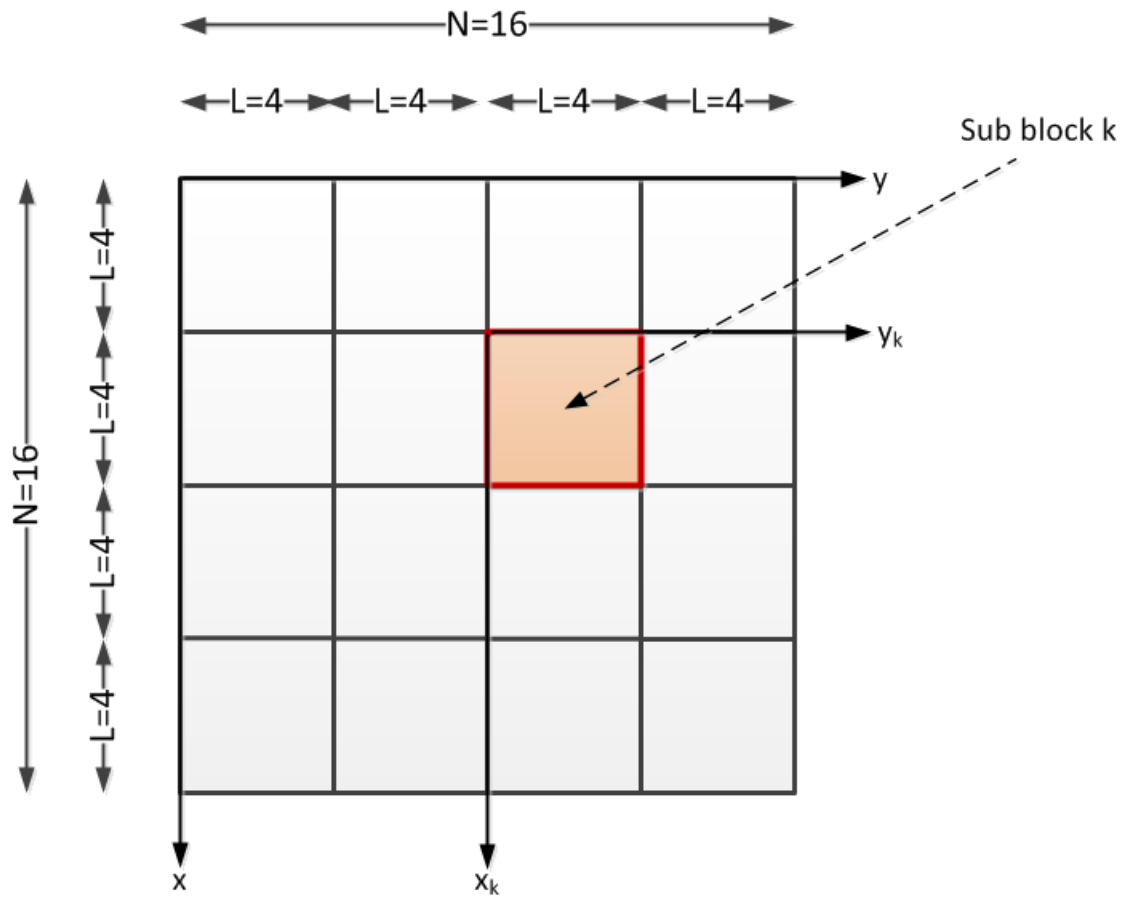


Figure 3.1 Macroblock decomposition into sub-blocks.

The topology of the multi-agent system is represented by a non-directed neighborhood graph as shown in Fig. 3.2. N_k represents the set of neighbors of agent$k$. This topology graph also

represents the network communication graph where each agent communicates only with its neighbors.

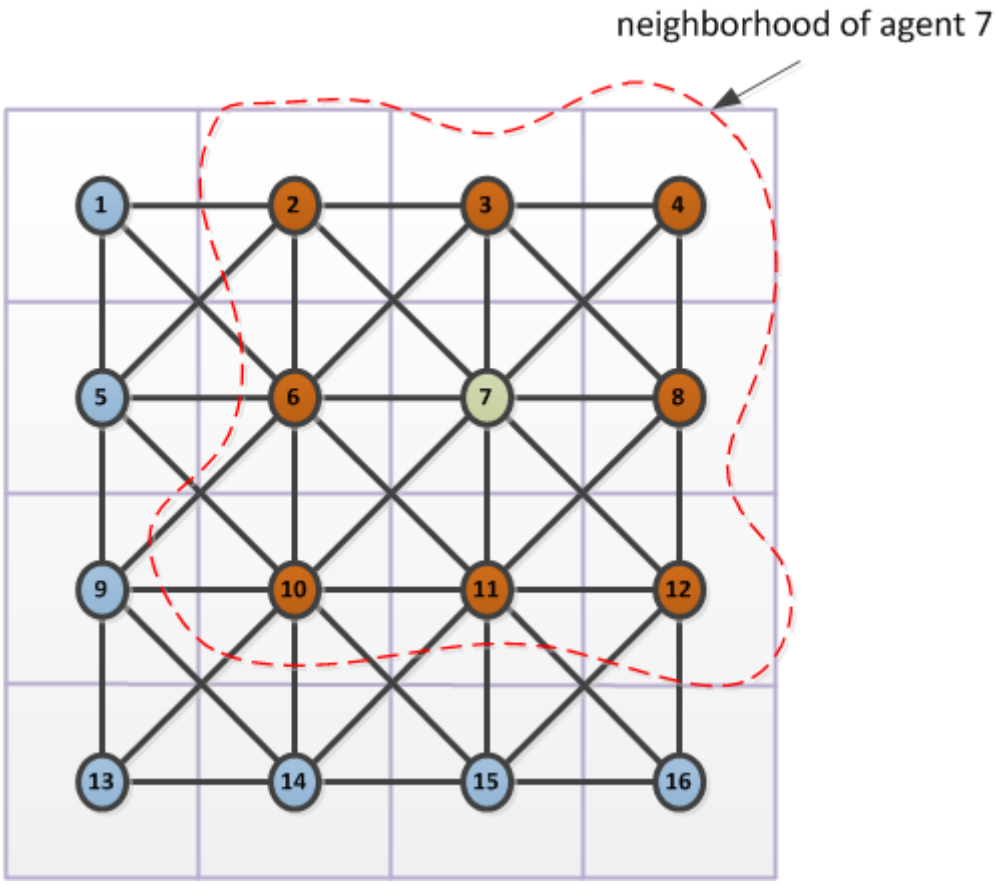


Figure 3.2 Neighborhood graph of agents.

3.2.1.3 Utility Function

$J_k(\hat{u}, \hat{v})$ cannot be used as a utility function for agent k because it doesn't depend on the action profile of the other agents. In order to introduce such dependency, the utility function of agent k can be chosen as:

$$U_k(\hat{u}, \hat{v}) = J_k(\hat{u}, \hat{v}) + \alpha * \sum_{i \in N_k}((\hat{u} - u_i)^2 + (\hat{v} - v_i)^2)^{\frac{1}{2}}. \quad (3.3)$$

The utility function includes a regularization term which is the Euclidean distance to the

motion vectors of the neighboring subblocks. In other words, the objective of a player is not only to minimize the SAD of its subblock but also to find a motion vector that is in high correlation with the motion vectors of the neighboring subblocks.

3.2.1.4 Action Set

The action set of agent$k$ is the set of motion vectors $(\hat{u}, \hat{v})$ within a specified search window.

3.2.2 Modeling the Problem as an Exact Potential Game

Definition of Potential Games [80]:

Player action sets $\{A_i\}_{i=1}^{n}$ *together with player objective functions* $\{U_i: A \to \mathbb{R}\}_{i=1}^{n}$ *constitute a potential functions, constitute a potential game if, for some potential function* φ: $U_i(a_i{}^1, a_{-i}) - U_i(a_i{}^2, a_{-i}) = \varphi(a_i{}^1, a_{-i}) - \varphi(a_i{}^2, a_{-i})$,

for every player $P_i \in P$ *, for every* $a_i^1, a_i^2 \in A_i$, *and for every* $a_{-i} \in A_{j \neq i}$.

A potential game, as previously defined, requires perfect alignment between the global objective and the players' local objective functions in the following sense: If a player unilaterally changed its action, the change in its objective function would be equal to the change in the potential function.

The proposed multi-agent block motion estimation problem can be modeled as a potential game by appropriately defining the players' utilities. First, we establish a global objective function that captures the notion of consensus. Next, we show that local objective functions can be assigned to each player, so that the resulting game is, in fact, a potential game.

Consider a consensus problem with n-player set P, where each player $P_i \in P$ has a finite action set A_i. A player's action set could represent the finite set of locations that a player could select. We will consider the following potential function for the consensus problem:

$$\varphi(a) = \sum_{i=1}^{n} J_i(a_i) + \sum_{P_i \in P} \sum_{P_j \in N_i} \frac{\|a_i - a_j\|}{2}, \quad (3.4)$$

where

$J_i(a_i)$ is the local cost function (SAD) of subblocki which is assigned to player i. This cost function depends only on the action a_i (motion vector) of player i.

N_i is the neighbor set of player i.

Now, the goal is to assign each player an objective function that is perfectly aligned with the global objective:

$$U_i(a_i, a_{-i}) = J_i(a_i) + \sum_{P_j \in N_i} \|a_i - a_j\|. \quad (3.5)$$

The utility function includes a term which is the distance to the motion vectors of the neighboring subblocks. In other words, the objective of a player is not only to minimize the SAD of its subblock but also to find a motion vector that is in high correlation with the motion vectors of the neighboring subblocks. Now, each player's objective function is only dependent on the actions of its neighbors.

Claim: Player objective functions (3.5) constitute a potential game with potential function (3.4), provided that the time invariant interaction graph induced by neighbor sets $\{N_i\}_{i=1}^{n}$isundirected, i.e.,

$$P_i \in N_j \Leftrightarrow P_j \in N_i$$

Proof: Since the interaction graph is time invariant and undirected, the potential function can be expressed as

$$\varphi(a) = J_i(a_i) + \sum_{k=1,k\neq i}^{n} J_k(a_k) + \sum_{P_j \in N_i} \|a_i - a_j\| + \sum_{P_k \neq P_i} \sum_{P_j \in N_k \setminus P_i} \frac{\|a_k - a_j\|}{2} \quad (3.6)$$

The change in the objective function of player P_i by switching from action $a_i{}^1$ to action $a_i{}^2$, provided that all other players collectively play a_{-i}, is

$$U_i(a_i{}^1, a_{-i}) - U_i(a_i{}^2, a_{-i}) = J_i(a_i{}^1) - J_i(a_i{}^2) + \sum_{P_j \in N_i} \|a_i{}^1 - a_j\| - \sum_{P_j \in N_i} \|a_i{}^2 - a_j\| =$$

$$\varphi(a_i{}^1, a_{-i}) - \varphi(a_i{}^2, a_{-i}) \quad (3.7)$$

This is an exact potential game.

3.2.3 Learning Algorithm

A fundamental solution concept for strategic form games is the Nash equilibrium:

Definition of Nash Equilibrium [81]: A strategy profile $a^* = (a_i^*, a_{-i}^*) \in \mathcal{A}$ is a pure-strategy *Nash equilibrium* (or simply a Nash equilibrium) of a game $(I, (\mathcal{A}_i), (u_i))$ if $u_i(a_i^*, a_{-i}^*) \geq u_i(a_i, a_{-i}^*)$, for every $i \in I$ and $a_i \in \mathcal{A}_i$; equivalently, $a_i^* \in BR_i(a_{-i}^*)$ for every $i \in I$. That is, a_i^* is a solution to the optimization problem $max_{a_i \in \mathcal{A}_i} u_i(a_i, a_{-i}^*)$.

At the Nash equilibrium, no player can improve his/her payoff by adopting a different strategy *unilaterally*; thus, no player has an incentive to unilaterally deviate from the equilibrium. The Nash equilibrium is a proper solution concept; however, the existence of a pure-strategy Nash equilibrium is not necessarily guaranteed. [81]

The most important property of potential games is *acyclicity*, which is also referred to as the finite improvement property (FIP).

Definition of Finite improvement property [72]:

A path in $(\mathfrak{T}, (\mathcal{A}_i), (u_i))$ is a sequence $(a[0], a[1], \dots)$ such that for every integer $k \geq 1$

, there exists a unique player $i$such that$a_i[k] \neq a_i[k-1] \in \mathcal{A}_i$ while$a_{-i}[k] = a_{-i}[k-1]$. $(a[0], a[1], \dots)$ is *an improvement path* if, for every$k \geq 1$,$u_i(a[k]) > u_i(a[k-1])$ where$i$ is the unique deviator at step$k$,$(\mathfrak{I}, (\mathcal{A}_i), (u_i))$ has the *finite improvement property (FIP)* if every improvement path is finite.

Theorem 1 ([72]): Every OPG with finite strategy sets has the FIP [72, Lemma 2.3]; that is, unilateral improvement dynamics are guaranteed to converge to a Nash equilibrium in a finite number of steps.

This potential game formulation provides a valuable theoretical framework for the proposed distributed multi-agent BME problem. First, the existence of a Nash equilibrium in potential games is guaranteed in many practical situations [72], but is not guaranteed for general strategic form games. Second, from the definition of a potential game, the Nash equilibrium for every local cost function is consistent with the global objective. Unilateral improvement dynamics in potential games with finite strategy sets are guaranteed to converge to the Nash equilibrium in a finite number of steps, i.e., they do not cycle [72]. Moreover, in finite player potential games, all equilibria are local maximizers of potential; since better reply adjustment processes increase potential, all equilibria are locally stable. As a result, learning algorithms can be systematically designed. The potential game framework, therefore, provides distributed optimization problems with theoretical support for problem simplification.

A variety of learning algorithms are available to facilitate the convergence of potential games to Nash equilibrium, e.g., best response, fictitious play, reinforcement learning, and spatial adaptive play. Best Response dynamics is a perfect fit for potential games.

3.2.3.1 Best Response Dynamics

The BRD can be formulated for a game with an arbitrary number of players. In its most used form, BRD operates in a sequential manner (sequential BRD) such that players update their actions in a round-robin manner. Within round *t+1* (with $t \geq 1$), the action chosen by player $k \in \kappa$ is computed as:

$$a_k(t+1) \in BR_k[a_1(t+1), \dots, a_{k-1}(t+1), a_{k+1}(t), \dots, a_K(t)]. \quad (3.8)$$

An alternative version of the BRD operates in a simultaneous way meaning that all players update their actions simultaneously [77]:

$$a_k(t+1) \in BR_k[a_{-k}(t)]. \quad (3.9)$$

Table 3.1 BRD Algorithm

Set t=0
Initialize $a_k(0) \in S_k$ for all players $k \epsilon \kappa$ (e.g. using random initialization)
Repeat
For $k = 1\ to\ K$ **do**
Update $a_k(t+1)$ using (3.8) or (3.9)
End for
Update $t = t + 1$
Until $\lvert a_k(t) - a_k(t-1) \rvert \leq \varepsilon$ for all $k \epsilon \kappa$

Theorem 2 ([76]): In potential and supermodular games, the sequential BRD converges to a pure NE with probability one.

The pseudo code of BRD for both instances is given in table 3.1. Convergence means that the distance between two successive action profiles remains below a certain threshold $\varepsilon > 0$.

When it converges, convergence points are typically pure NE [76]. There are no convergence results for general games using BRD. However, when exact potential games are considered, then there exist sufficient conditions under which the convergence of the sequential BRD to a pure NE is always guaranteed [76]. On the other hand, unlike the sequential BRD, there does not seem to exist general results that guarantee the convergence of the simultaneous BRD. [77]

3.3 Proposed Distributed Block Motion Estimation Scheme

So far, the BME problem is formulated in a game-theoretic multi-agent setting and is then modeled as an exact potential game. The solution methodology proposed to find the NE of the game is based on BRD where, in each round, each agent autonomously tries to optimize its local utility function in (3.3) given the actions of the other agents. In order to carry out this optimization step, agents should be equipped with local optimization capabilities. Due to the non-convexity of the agent's utility function (3.3), we resort to modern optimization techniques. Particle Swarm Optimization (PSO) is chosen as the global optimization algorithm used by the agents due to its profound intelligence and simple algorithmic structure. Each agent is equipped with a PSO engine that finds its best response at each time step given the actions of the other players by optimizing its local utility function.

3.3.1 Description of the proposed distributed algorithms

The proposed algorithm solves the problem of BME through game-theoretic interactions among a network of self-interested, distributed computational agents. The macroblock is divided into subblocks and an agent is defined for each subblock. Then, a swarm of PSO particles is defined for each agent to serve as its local processing engine. According to the presented game

theoretic framework, agents perform BRD that drives the agents towards consensus on the common MV of the whole MB which is also the NE of the underlying potential game. Two versions of this approach are presented: a sequential version that uses sequential BRD, and a simultaneous version that uses simultaneous BRD. The main distinction between these two algorithms lies in the mode and frequency of the inter-agent communication needed during the process to update neighborhood information which also determines the level of dependency between the agents.

3.3.1.1 Sequential algorithm

In this algorithm, agents use sequential BRD. In each round, agents update their actions in a synchronous round robin fashion, from agent 1 until agent K in each round, according to (3.8). At round t+1, agent k needs to find its BR or utility maximizing action given by:

$$a_k(t+1) = \arg max_{a_k} u_k\big(a_k, a_1(t+1), \dots, a_{k-1}(t+1), a_{k+1}(t), \dots, a_K(t)\big). \quad (3.10)$$

According to our definition of the agent's utility function in (3.3), the utility function of an agent depends only on the actions of its neighboring agents defined in N_k. Therefore, an agent needs updated information about the actions of its causal neighbors before calculating its best response. Therefore, in each round of the sequential algorithm, we iterate over the agents and each agent performs a communication step followed by an optimization step to find its BR at this round.

3.3.1.1.1 Communication step:

This step allows each agent to receive up-to-date information about the actions of its causal neighbors through inter-agent communication. This is shown in Fig. 3.3.

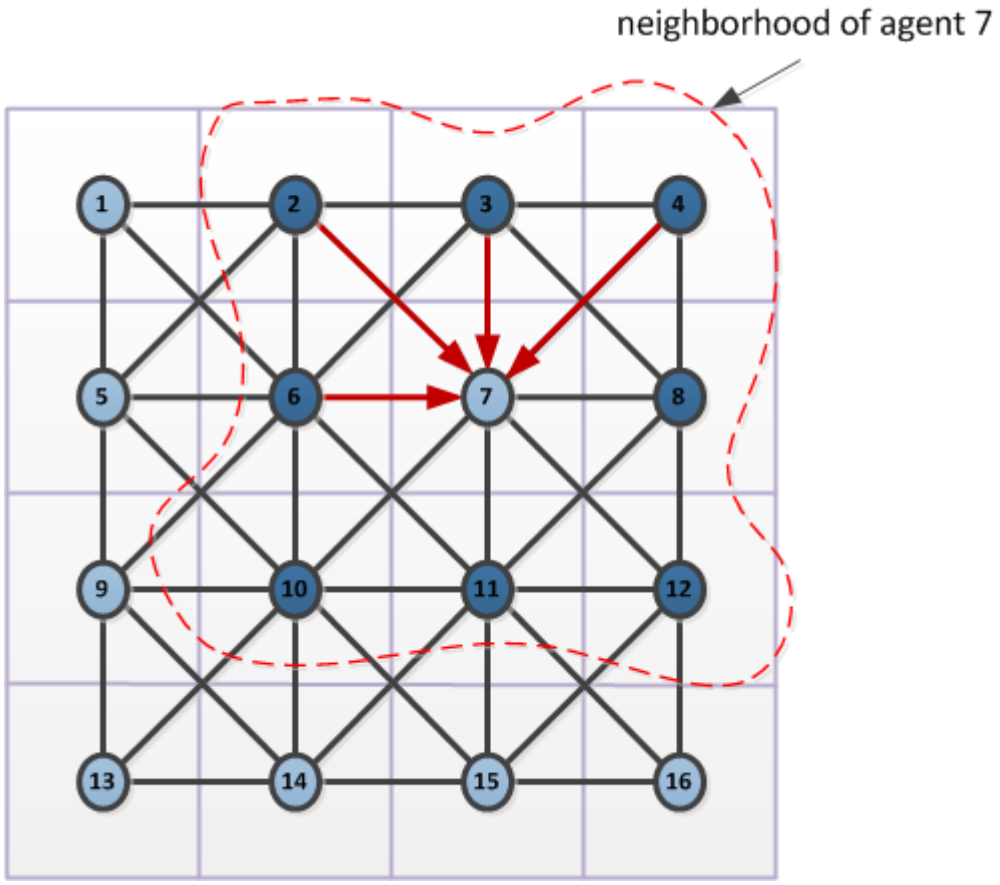


Figure 3.3 Communication step in the sequential algorithm where agent k receives updated information about the actions of the causal agents in its neighboring set.

3.3.1.1.2 Local Optimization Step

In this step, an agent runs a set of N_t PSO iterations to find its best response (BR) or its best action at time t given the updated actions of its neighbors. This is done by optimizing the utility function of the agent as given in (3.5) and using it as the fitness function in the PSO process.

A swarm of M particles is first defined for each agent. This is done by first initializing a set of particles for the given MB and then randomly selecting from this set M particles for each agent. This is done as follows. Based on the assumption that the motion field is smooth and varies slowly, there are strong spatial correlations between motion vectors of neighboring blocks within the same frame as well as strong temporal correlations with motion vectors of blocks in the

previous frame. Typically, a macroblock (MB) has 8 immediate neighbors. For a raster-search order, the available apriori information for the current block are the motion vectors of only the four causal neighboring blocks within the same frame that have already been found before block matching for the current MB is conducted. To exploit the existing spatial correlation, motion vectors of these four neighbors are used to initialize the positions of four swarm particles in the search area. Additionally, the current block has apriori information of the motion vectors of the collocated MB and its neighbors from the previous frame. In order to exploit the existing temporal correlation, we initialize five swarm particles to the motion vectors of the collocated MB in the previous frame and four of its direct neighbors as shown in Fig. 3.4. We also initialize one of the particles to the (0, 0) motion vector (MV) to account for static blocks. Therefore, for an MB at location (i,j) in frame t, we initialize a pool of 10 particles as follows:

$$\{x_1, x_2, x_3, \dots, x_{10}\} =$$

$$\{Mv^{t}_{i-1,j-1}, Mv^{t}_{i-1,j}, Mv^{t}_{i-1,j+1}, Mv^{t}_{i,j-1}, Mv^{t-1}_{i,j}, Mv^{t-1}_{i,j+1}, Mv^{t-1}_{i+1,j-1}, Mv^{t-1}_{i+1,j}, Mv^{t-1}_{i+1,j+1}, (0,0)\}. \quad (3.11)$$

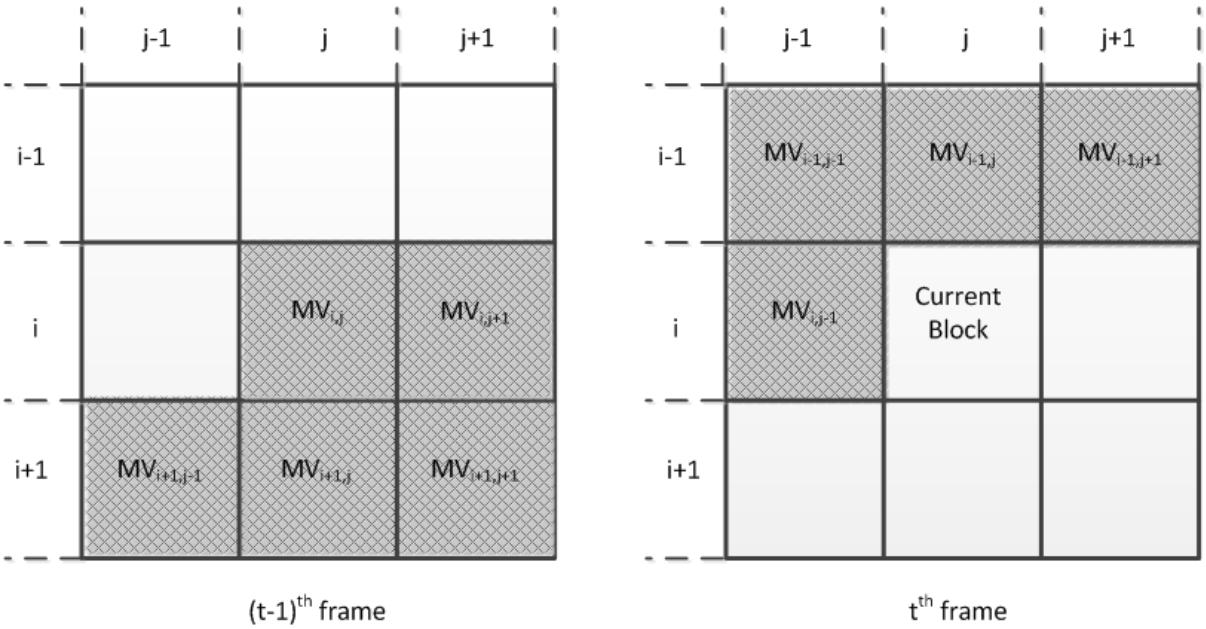


Figure 3.4 Initialization of particles positions of a given MB

Then, for each agent within each MB, a set of M particles are randomly chosen from the initialized pool of particles of the corresponding MB.

In order to decrease the computational complexity of this stage, fitness function history preservation proposed in Chapter 2 is adopted to avoid unnecessary redundant fitness function calculations of search points that have been visited before by any particle during the PSO process. In addition, the maximum velocity which limits the flying speed of the particles is adaptively changed in this PSO process as was proposed in Chapter 2. Therefore, in this modified PSO algorithm, a higher v_{max} value is adopted in the early stage of the search process and a lower value later to perform a local search. A linearly decreasing function is adopted to gradually reduce the v_{max} value in the current iteration in proportion to the iteration number, this is given by:

$$v_{max}(j) = \frac{V_{max}}{j}, \tag{3.12}$$

where $V_{max}$is an empirically determined value and j is the iteration number.

This PSO process terminates whenever the maximum number of iterations N_t is reached. Early termination of the search is allowed whenever the fitness value of the global best position is less than a predefined threshold value T_{th} and when the fitness value associated with the global best position remains the same for K_{max} iterations, even if the maximum iteration number N_t is not yet reached. The pseudo code of the sequential algorithm is shown in Table 3.2.

Table 3.2 Pseudocode of the proposed sequential ME algorithm

For each MB in the frame **do**
Initialize a set of 10 particles according to (3.11) and as shown in Fig. 3.4
For each agent or subblock k=1,…,K **do**
Define a swarm of M PSO particles
Initialize the fitness history array entries to zeros
Initialize particle velocities to zeros
Initialize particle positions by randomly selecting M particles from the MB particle set
End for
BR Rounds:
Repeat
For each agent or subblock k=1,…,K **do**
Communication Step:
Update the matrix of neighbors's actions or global best positions as shown in Fig. 3.3
BR Optimization Step Using PSO:
Repeat
For each particle i=1,…,M **do**
Check its flag in agent k history array
If the flag is 0 then
Calculate utility function of the agent
Update the particle's best position P_i and the global best position P_i of the agent
Save the value of the fitness value in the history array
Set flag to 1
Else
Retrieve the value of the fitness function from the history array
Update P_i
End if
Adaptively change v_{max} using (3.12)
Update the velocity
Update the position
End for
Until stopping conditions of the PSO process are met
End for
Until all the players' strategies become stationary or the prescribed maximum number of BR rounds T is reached.
End for

3.3.1.2 Simultaneous Algorithm

The proposed simultaneous algorithm uses simultaneous BRD which operates in a simultaneous way meaning that, in each round, all agents update their actions simultaneously according to:

$$a_k(t+1) = \arg max_{a_k} u_k\left(a_{-k}(t)\right). \quad (3.13)$$

That is, to find the BR of agent k at round $t+1$, we need information about the actions of its neighbors in the previous round t. this offers the important advantage of breaking the

dependency between the agents within each round. The steps of the proposed algorithm are as follows. The MB is divided into subblocks as shown in Fig. 3.1 and an agent is defined for each subblock. A Swarm of M PSO particles is defined and initialized for each agent. Several rounds of simultaneous BRD are carried out for the subblocks. For each round, an inter-agent communication step is first performed. Then, we iterate over the agents to calculate their BR through a local optimization process using PSO.

3.3.1.2.1 Communication Step

At the beginning of each round, all agents need updated information about the actions of their neighbors in the previous round in order to find their BR at the current round. A broadcast step is performed where each all agents broadcast to their neighbors their actions attained from the previous round. This is shown in Fig. 3.5.

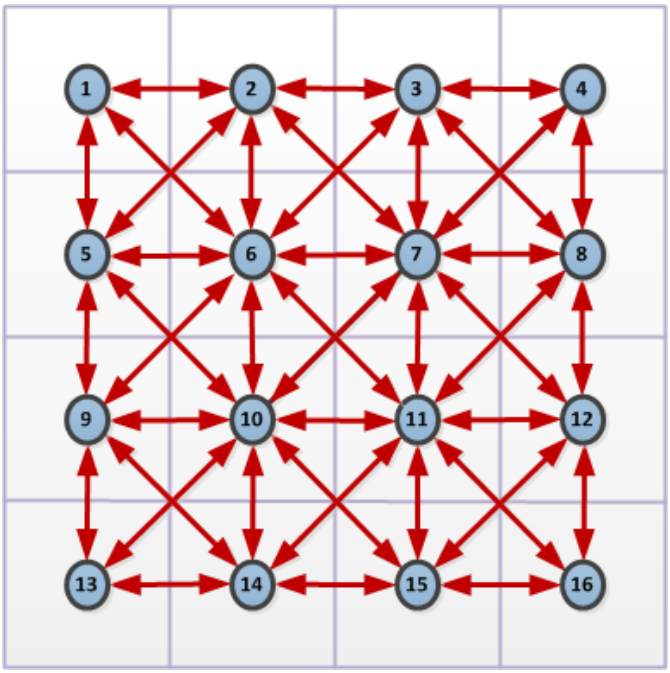


Figure 3.5 Communication step in the simultaneous algorithm where each agent broadcasts information about its current action to the agents in its neighboring set.

3.3.1.2.2 Local Optimization Step

After updating the actions of their neighbors, agents use the modified PSO optimization process to find their best response (BR) according to (3.13).

3.3.1.2.3 Final MV Search Step

Unlike the sequential BRD, there does not seem to exist general theoretical results that guarantee the convergence of the simultaneous BRD [27]. Therefore, in the proposed simultaneous ME algorithm, the maximum number of simultaneous BR rounds T might be reached before convergence of the MVs of all the agents to a common MV which is the minimizer of the potential function. In that case, a final MV search step is needed to choose, from the agents' estimated MVs, the MV that minimizes the SAD of the whole MB. This is done efficiently by approximating the SAD of the whole MB as the sum of SAD values of its subblocks. The SAD values for the subblocks are then calculated for each of the estimated MVs of the players. The fitness function history preservation property of the subblocks is utilized here to reduce the computational complexity of this step. The pseudo code of the sequential algorithm is shown in Table 3.3.

Table 3.3 Pseudo code of the proposed simultaneous ME algorithm

For each MB in the frame **do**
Initialize a set of 10 particles according to (3.11) and as shown in Fig. 3.4
For each agent or subblock k=1,…,K **do**
Define a swarm of M PSO particles
Initialize the fitness history array entries to zeros
Initialize particle velocities to zeros
Initialize particle positions by randomly selecting M particles from the MB particle set
End for
BRD Rounds:
Repeat
Communication Step:
All Agents broadcast their actions to their neighbors.
All agents receive information from their neighbors about their actions
For each agent or subblock k=1,…,K **do**
BR Optimization Step Using PSO:
Repeat
For each particle i=1,…,M **do**
Check its flag in agent k history array
If the flag is 0 then
Calculate utility function of the agent
Update the particle's best position P_i and the global best position P_i of the agent
Save the value of the fitness value in the history array
Set flag to 1
Else
Retrieve the value of the fitness function from the history array
Update P_i
End if
Adaptively change v_{max} using (3.12)
Update the velocity
Update the position
End for
Until stopping conditions of the PSO process are met
End for
Until all the players' strategies become stationary or the prescribed maximum number of BR rounds T is reached.
If *convergence* is not reached then
Final MV search step:
End If
End for

3.3.1.3 Comparison between the Proposed Sequential and Simultaneous Algorithms

The two proposed algorithms describe how the problem of block ME can be solved in a decentralized manner by a system of autonomous agents which can, in a distributed fashion, optimize a global objective function. By means of non-cooperative games and local communication, each (selfish) agent tries to optimize its own utility leading eventually to an

emergent global welfare. The two proposed schemes, however, are different in several aspects. First, the sequential algorithm uses sequential BRD which offers a positive theoretical guarantee of convergence to the NE of the potential game. On the other hand, the presented simultaneous scheme is based on simultaneous BRD, which doesn't possess any theoretical guarantee of convergence. When several players move simultaneously, the potential may not be increasing and then the convergence of BRA to a NE of the game is not guaranteed [82].Nevertheless, unlike the sequential algorithm which requires continuous inter-agent communication, the presented simultaneous scheme eliminates data dependencies between the agents within each round and limits inter-agent communication to a single step at the beginning of each round. Therefore, agents are allowed to update their actions simultaneously. This offers the important advantage of parallelism at the agents' level and thus makes the simultaneous algorithm amenable to parallel implementations.

3.3.1.4 Computational Complexity Analysis

In block matching motion estimation, the average number of fitness function evaluations for each MB is used as a metric of the computational complexity. In the literature, the fitness function usually used for block motion estimation is the SAD defined in (3.1). For each MB of size of 16X16, there are 256 subtractions and 255 additions that are to be carried out in calculating the Sum of Absolute Difference (SAD). Hence, the computing complexity is $O(N^2)$, where N is the number rows or columns in NxN square MB. In the proposed simultaneous and sequential algorithms, each MB is divided into K subblocks as shown in Fig. 3.1. The cost function of agent k is the SAD of the subblock of dimension LxL as given in (3.2). Therefore, each SAD computation performed by an agent is $O(L^2)$. Since, as illustrated in Fig.2, $N^2/L^2 = K$, then we

can say that each cost function computed by an agent corresponds to $1/K$ MB fitness function evaluation.In the proposed algorithms, we have K agents within a given MB. Agents will go through a maximum of$T$ BR rounds. In each round, the M particles of each agent will perform a maximum of N_t PSO iterations. Therefore, the maximum total computational complexity of the proposed decentralized algorithms, in terms of fitness function evaluations per MB, is given by:

$$Computational\ Complexity_{Decentralized} = K * \frac{T*N_t*M}{K} = T * N_t * M. \quad (3.14)$$

Notice that the computational complexity per MB of a centralized PSO-based block ME algorithm [10] where a single PSO swarm of M'particles is defined for an MB with a total number of iterations N_t' is given by:

$$Computational\ Complexity_{Centralized} = N_t' * M'. \quad (3.15)$$

From (3.14) and (3.15), we notice that if the parameters$T$, N_t, and M are chosen properly, the computational complexity of the proposed distributed algorithms can be made equivalent to that of a centralized PSO-based ME algorithm.

3.4 Parallel implementation

In this section, a multicore implementation of the proposed simultaneous algorithm is proposed using the MATLAB® Parallel Computing Toolbox™ (PCT).

From the pseudo code of the simultaneous algorithm given in Table 3.3, we notice that the proposed ME algorithm for each MB in the frame is made up of the following steps: Initialization step to initialize the PSO swarms for the agents followed by the simultaneous BR rounds. The initialization step for each agent is completely independent from that of the other agents. Therefore, it can be parallelized. Moreover, within each round of the simultaneous BR process, agents update their actions using PSO simultaneously and independently. Therefore, this local

PSO optimization step is also parallelizable across the agents.

The details of the proposed parallel implementation are shown in table and explained as follows:

3.4.1 Parallel agents processing using Looping over a Distributed Range (for-drange)

In our implementation, the MB is partitioned equally along the subblocks or agents among the different *labs*. Therefore, we have used the *for-drange* construct to allow the simultaneous processing of each lab of its assigned agents in the MB.

3.4.2 SPMD block

This allows us to implement data-parallelism where each Matlab *lab* executes the same lines of code but on different subblocks of a given MB. Within the *spmd* block, communication or synchronization is allowed between labs. In the proposed parallel implementation, an *spmd* block is started for each MB in the frame as shown in Table 3.4.

3.4.3 Inter-agent communication using labSendReceive

During the communication step the proposed simultaneous algorithm, each agent, or subblock, is required to broadcast to its neighboring agents the global best position, P_g, found so far in the optimization process. Each agent will also receive from each of its neighbors the value of the global best position acquired by that neighbor. This interlab communication within the parallel job is implemented using *labSendReceive*.

Table 3.4 Pseudo code of the parallel implementation of the proposed simultaneous me algorithm using Matlab

For each MB in the frame **do**
Initialize a set of 10 particles according to (3.11) and as shown in Fig. 3.4
Spmd
For k=***drange*** (1,…,K) **do**
Define a swarm of M PSO particles
Initialize the fitness history array entries to zeros
Initialize particle velocities to zeros
Initialize particle positions by randomly selecting M particles from the MB particle set
End *for drange*
BRD Rounds:
Repeat
Communication Step:
All Agents exchange information with their neighbors using ***labSendReceive***
For k=***drange*** (1,…,K) **do**
BR Optimization Step Using PSO:
Repeat
For each particle i=1,…,M **do**
Check its flag in agent k history array
If the flag is 0 then
Calculate utility function of the agent
Update the particle's best position P_i and the global best position P_i of the agent
Save the value of the fitness value in the history array
Set flag to 1
Else
Retrieve the value of the fitness function from the history array
Update P_i
End if
Adaptively change v_{max} using (3.12)
Update the velocity
Update the position
End for
Until stopping conditions of the PSO process are met
End *for drange*
Until all the players' strategies become stationary or the prescribed maximum number of BR rounds T is reached.
End *spmd*
If *convergence* is not reached then
Final MV search step:
End If
End for

3.5 Simulation Results

3.5.1 Experimental setup

Several test video sequences of various formats and various motion intensity, (QCIF: 176x144), LD (CIF: 352x288), SD (480p: 832x480), and HD (720p: 1280x720) downloaded from

[68, 69], have been used to test the performance of our proposed algorithm and compare it to existing techniques. Results are presented with two distinct criteria: computational complexity and objective motion estimation quality.

The proposed algorithm is simulated on a server with two Intel® Xeon® E5520 2.66GHz CPUs (total of 8 physical CPU cores equivalent to 16 logical cores due to the hyper-threading property of Intel CPUs) and 32GB RAM. The execution platform is Matlab R2012a.

In our simulations, every frame is divided into MBs of size 16 * 16 pixels. The search step-size is one integer pixel and we used one reference frame which is the previous frame. The search parameter p which defines the search area is chosen to be 15 for all the tested sequences except for the HD (Parkrun) video sequence where p was chosen to be 31.

3.5.2 Simulation parameters

For the PSO algorithm, a pool of 10 particles is initialized for each MB according to (3.11). Form this pool, a set of M particles are randomly selected for each player within the MB. As discussed in section 3.1.4, the computational complexity of the algorithm depends on the parameters T, M, and N_t. These parameters should be chosen properly so that the computational complexity of the proposed distributed algorithms can be made equivalent to that of a centralized PSO-based ME algorithm [52]. In [52], a swarm of 9 particles ($M' = 9$) are used for a total of 5 iterations ($N'_t = 5, K'_{max} = 2$) which requires a maximum of 45 fitness function evaluations per MB. In the proposed algorithms, the number of particles per subblock is chosen to be $M = 3$. Different values of N_t and K_{max} are chosen for the proposed sequential and simultaneous algorithms as follows. The proposed sequential algorithm is guaranteed to converge in T_{conv} number of BR rounds. According to our simulations, a good overall performance over a wide

range of video sequences is provided for a choice of N_t = 3(and K_{max} = 2) which is found to converge in an average number of BR rounds $T_{conv} \cong 5$ if the number of players is$K = 16$, and a choice of N_t = 5(and K_{max} = 3) which is found to converge in an average number of BR rounds $T_{conv} \cong 3$ if the number of players is$K = 4$. This requires a computational complexity of $T_{conv} * M * N_t = 45$fitness function evaluations, which is equivalent to that of the centralized PSO-based ME algorithm in [52]. On the other hand, as mentioned previously, the proposed simultaneous algorithm doesn't possess any guarantees for its convergence. According to our simulations, a good overall performance over a wide range of video sequences is obtained for a choice of N_t = 5(and K_{max} = 3) and a fixed number of BR rounds $T = 2$.The choice of the BR rounds $T = 2$ provides the important advantage of minimizing the communication overhead required in the proposed simultaneous algorithm since the number of communication steps needed between the players is only one in this case. The resulting maximum computational complexity in this case is $T * M * N_t = 2 * 3 * 5 = 30$ fitness function evaluations for the BR process in addition to the computations needed during the final MV search step. Overall, the computational complexity of the proposed simultaneous algorithm is almost equivalent to the centralized ME approach in [52].

The pre-set minimum MSE error, T_{th}, is another empirically determined threshold that can regulate the accuracy/complexity tradeoffs. If the threshold is too large, the algorithm tends to run fast at the cost of a lower accuracy. In our simulations, the threshold for MSE, T_{th}, was chosen to be 7. The maximum allowed velocity for the PSO particles, v_{max}, is dynamically varied in every iteration of the PSO process. It is initially set to $V_{max}$and then linearly decreased according to the iteration number. In our simulations, we chose V_{max} = 15, for a search range of ±15, and V_{max} = 31, for a search range of±31, as a starting value for the maximum allowed velocity which adaptively decreases in each iteration giving an average value of around 20% of the dynamic range in each

case.

3.5.3 Numerical analysis of convergence

In Section 3.2.3, we have provided the theoretical proofs of convergence for the proposed sequential ME algorithm. In this section, simulation results of convergence are provided. Table 3.5 provides an empirical convergence analysis of the sequential BRD algorithm interms of the average number of BR rounds needed for convergence, T_{conv}, for the different video sequences. We also analyze the effect of the number of players K on the convergence speed of the algorithm. Two values of K are considered. Setting $K = 4$ means that the MB is divided into 2x2 subblocks each of size 8x8 pixels, whereas setting $K = 16$ means that the MB is divided into 4x4 subblocks each of size 4x4 pixels. Table 3.5 shows that only a few iterations are needed to obtain convergence, which is a quite typical behavior for sequential BRD-type iterative procedures [76]. It is noticed that the convergence speed decreases with the increase in the number of players.

Table 3.5 Empirical Convergence Analysis of the sequential BRD algorithm in terms of the average number of BR rounds needed

Sequence	**Number of BR Rounds T_{conv} for $K = 16$**	**Number of BR Rounds T_{conv} for $K = 4$**
Soccer, QCIF, 15fps, p=15	5.9	3.54
Bus, QCIF, 15 fps, p=15	5.6	3.29
Soccer, CIF, 30fps, p=15	3.87	2.93
Bus, CIF, 30 fps, p=15	5.18	2.91
Tennis, CIF,30 fps, p=15	4.9	3.23
Stefan, CIF, 30fps, p=15	4.4	3.03
Foreman, CIF, 30fps, p=15	5.2	3.22
Container, CIF, 30fps, p=15	3	2.21
RaceHorses, 480p, 30 fps, p=15	4.8	3.88
Parkrun, 720p, 30 fps, p=31	3.7	2.6

3.5.4 Motion estimation quality

Objective motion estimation quality is measured interms of Peak Signal to Noise Ratio (PSNR) values averaged over the first 100 frames of each test video sequence.

First, we compare the performance of the proposed sequential and simultaneous ME algorithms, interms of motion estimation quality, for different values of the number of players K. As shown in Table 3.6, increasing the number of players K in the game results in an improvement in the motion estimation quality of both of the proposed algorithms. This can be explained as follows. As shown in Table 3.5, increasing the number of players leads to an increase in the needed BR rounds for convergence in the proposed sequential algorithm. Consequently, this leads to an increase in the number of search points explored and a more exhaustive search space exploration which prevents the algorithms from falling into local minima. Moreover, increasing the number of players K in the game means incorporating a larger number of computing agents to solve the ME problem which leads to a better estimation quality. Another important observation from the results shown in Table 3.6 is that, for the same value of K, the motion estimation quality of the proposed sequential algorithm exceeds that of the proposed simultaneous approach. The reason behind this lies in the fact that, unlike the proposed sequential algorithm, the simultaneous BRD used in the proposed simultaneous algorithm doesn't necessarily converge to the NE and the proposed final MV search step used provides only a suboptimal solution of the game.

Table 3.7 gives the average PSNR results for the ES algorithm and several traditional fast searching techniques, like TSS [16], 4SS [18], DS [19], and ARPS [22]. PSNR results are also given for the recently proposed Pattern Based PSO ME (PBPSO) algorithm given in [51] and the PSO ME algorithm proposed in [52]. The simulation results presented are based on the averages of the data (PSNR and search point) obtained from 50 repeated runs of the proposed algorithms to strengthen the statistical significance. Increasing the number of runs also yield only very

negligible changes to the averages which do not differ significantly. Simulation results show that both of the proposed algorithms provide an improvement in motion estimation quality as compared to the other techniques. Fig. 3.6, Fig. 3.7, Fig. 3.8, Fig. 3.9, Fig. 3.10, Fig. 3.11, Fig. 3.12, and Fig. 3.13 show that the proposed algorithms can closely follow the PSNR values of the ES method on the frame-by-frame basis.

Table 3.6 Motion estimation quality in terms of PSNR of the proposed sequential and simultaneous algorithms for different values of K

	Sequential Algorithm		**Simultaneous Algorithm**	
Sequence	$K = 4$	$K = 16$	$K = 4$	$K = 16$
Soccer, QCIF, 15fps, p=15	23.78	24.48	23.567	24.39
Bus, QCIF, 15 fps, p=15	22.74	22.99	22.594	22.941
Soccer, CIF, 30fps, p=15	29.311	29.57	28.9	29.49
Bus, CIF, 30 fps, p=15	25.126	25.36	24.92	25.28
Tennis, CIF,30 fps, p=15	28.185	28.58	27.89	28.495
Stefan, CIF, 30fps, p=15	25.86	26.7	25.645	26.54
Foreman, CIF, 30fps, p=15	33.629	34.228	33.46	34.18
Container, CIF, 30fps, p=15	32.1295	32.65	31.97	32.403
RaceHorses, 480p, 30 fps, p=15	28.251	28.94	28.034	28.87
Parkrun, 720p, 30 fps, p=31	25.4836	25.523	25.4188	25.508

Table 3.7 Motion estimation quality in terms of PSNR of the proposed Algorithms as compared to existing techniques

Sequence	**ES**	**TSS**	**4SS**	**DS**	**ARPS**	**PSO [52]**	**PBPSO [51]**	**Simultaneous Algorithm $K = 16$**	**Sequential Algorithm $K = 16$**
Soccer, QCIF, 15fps, p=15	25.01	24.02	22.10	23.26	23.761	24.33	20.12	24.39	24.48
Bus, QCIF, 15 fps, p=15	23.35	21.88	19.76	20.41	21.036	22.809	17.579	22.941	22.99
Soccer, CIF, 30fps, p=15	30.19	28.21	27.01	27.69	28.662	29. 329	21.847	29.49	29.57
Bus, CIF, 30 fps, p=15	25.60	22.37	19.78	20.33	21.793	24.925	18.448	25.28	25.36
Tennis, CIF,30 fps, p=15	29.19	26.85	27.76	28.12	28.070	28.224	24.304	28.495	28.58
Stefan, CIF, 30fps, p=15	26.93	24.62	23.70	23.98	26.038	26.482	20.238	26.54	26.7
Foreman, CIF, 30fps, p=15	34.68	33.49	33.80	34.24	34.185	34.174	31.257	34.18	34.228
Container, CIF, 30fps, p=15	32.84	26.87	23.59	23.54	29.11	32.386	18.573	32.403	32.65
RaceHorses, 480p, 30 fps, p=15	29.33	26.80	24.89	26.01	27.44	28.86	21.425	28.87	28.94
Parkrun, 720p, 30 fps, p=31	25.61	20.43	23.66	23.31	25.33	24.4	19.094	25.508	25.523

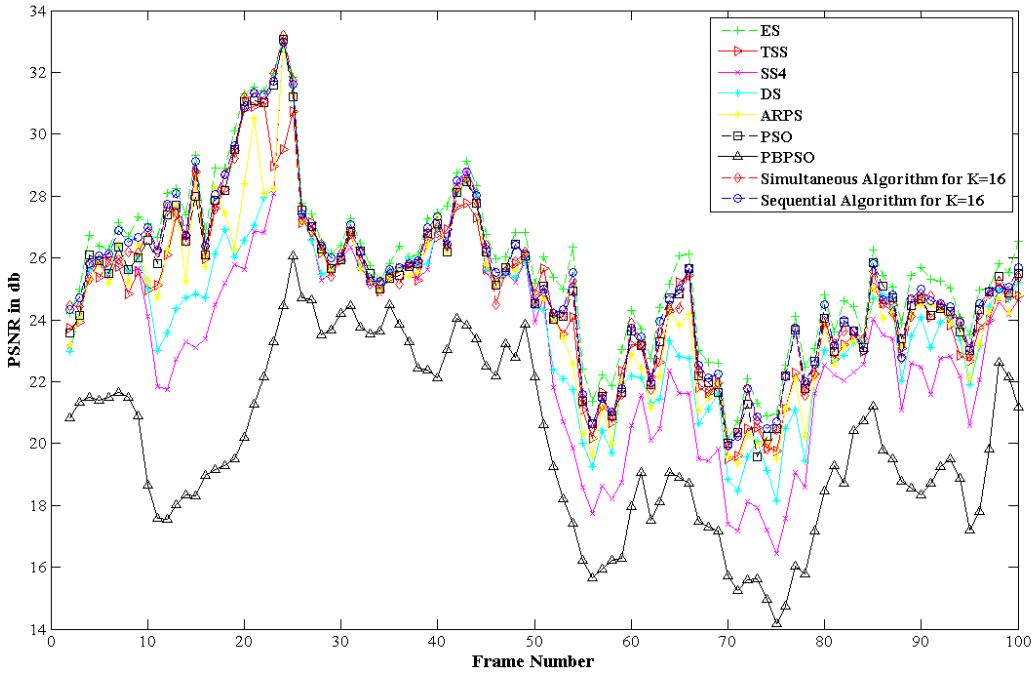


Figure 3.6 Motion estimation quality measured in PSNR for "Soccer, QCIF" sequence.

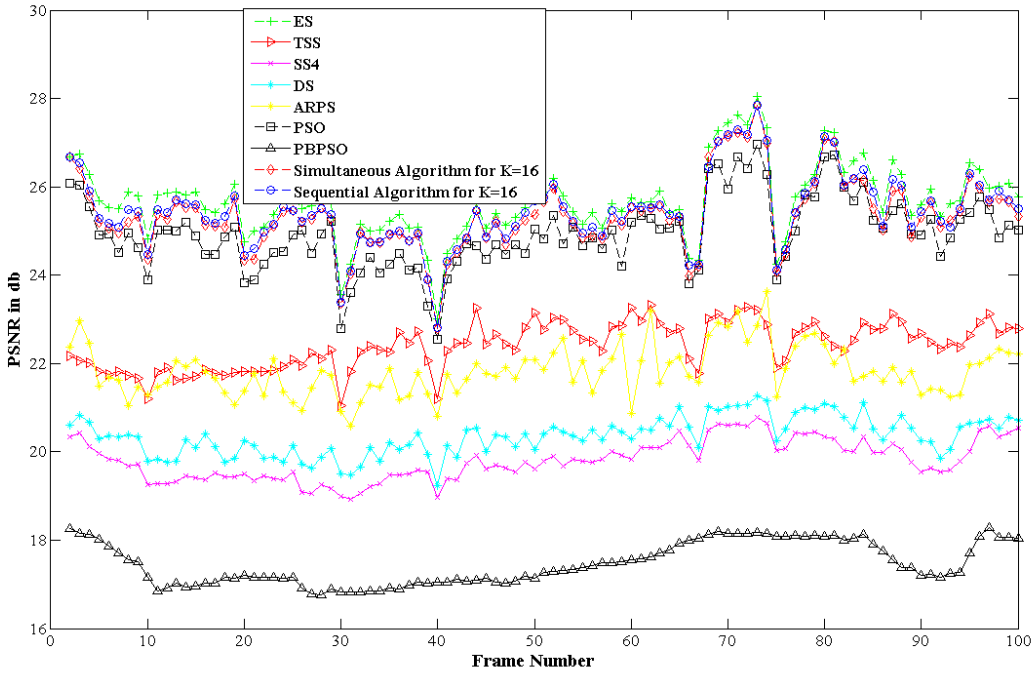


Figure 3.7 Motion estimation quality measured in PSNR for "Bus,CIF" sequence.

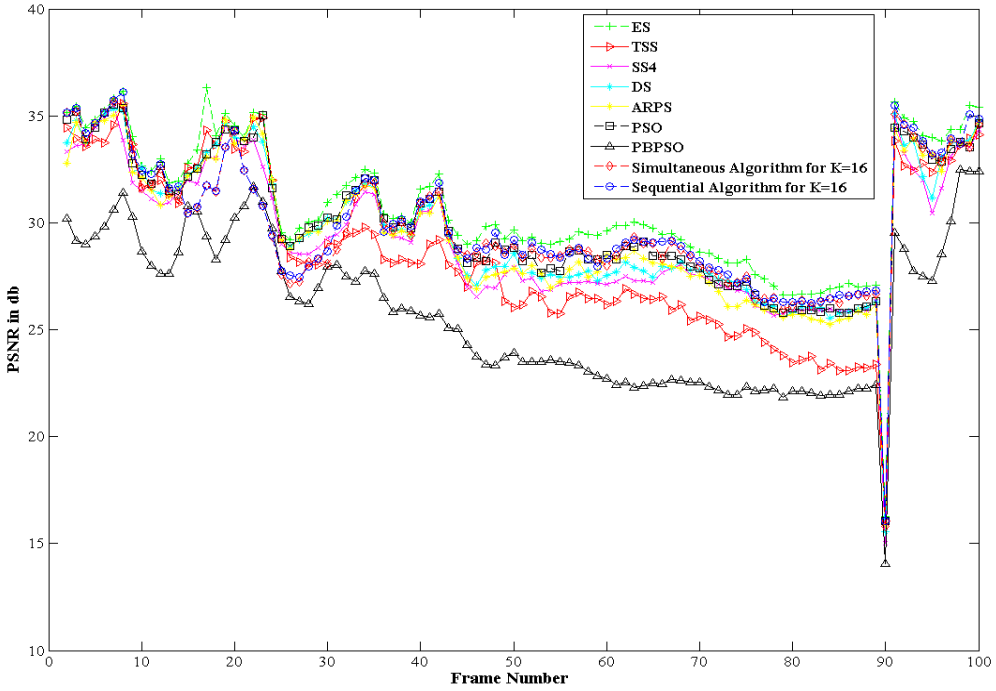


Figure 3.8 Motion estimation quality measured in PSNR for "Tennis, CIF" sequence.

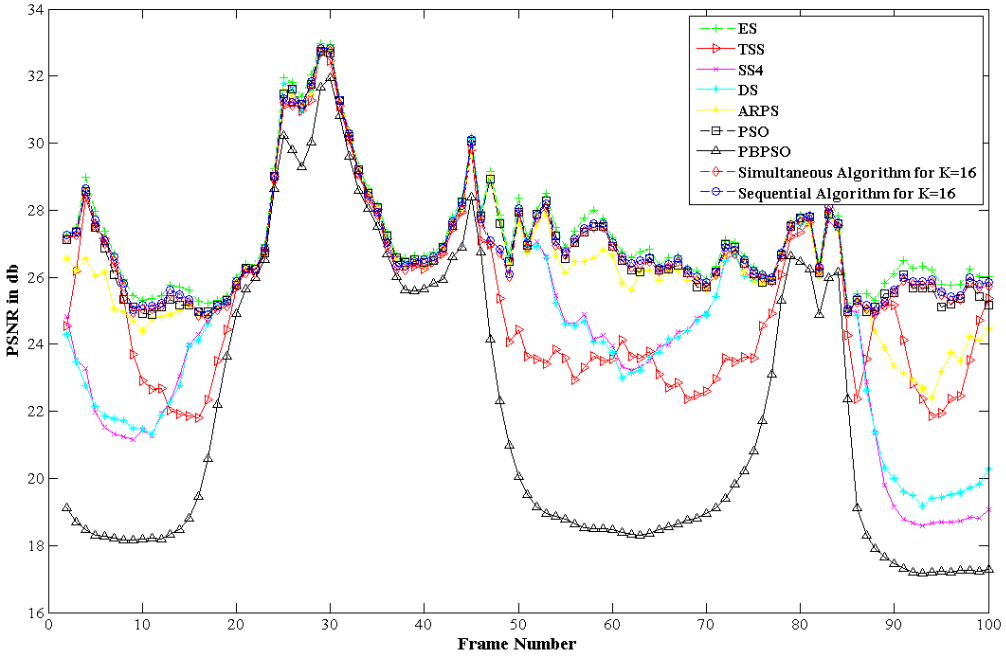


Figure 3.9 Motion estimation quality measured in PSNR for "Stefan, CIF" sequence

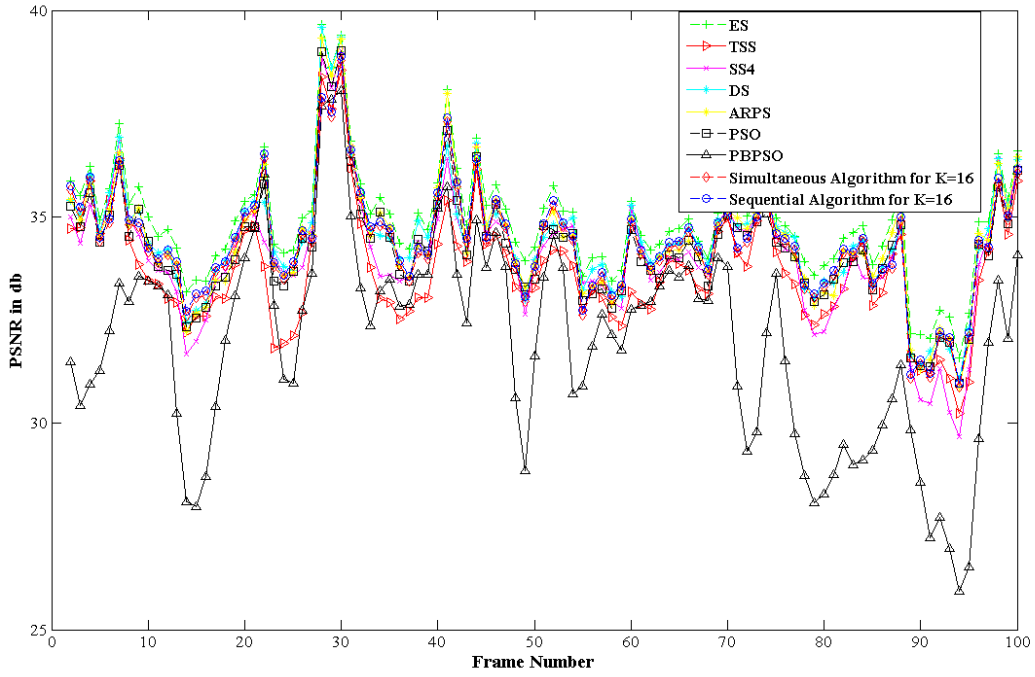


Figure 3.10 Motion estimation quality measured in PSNR for "Forman, CIF" sequence

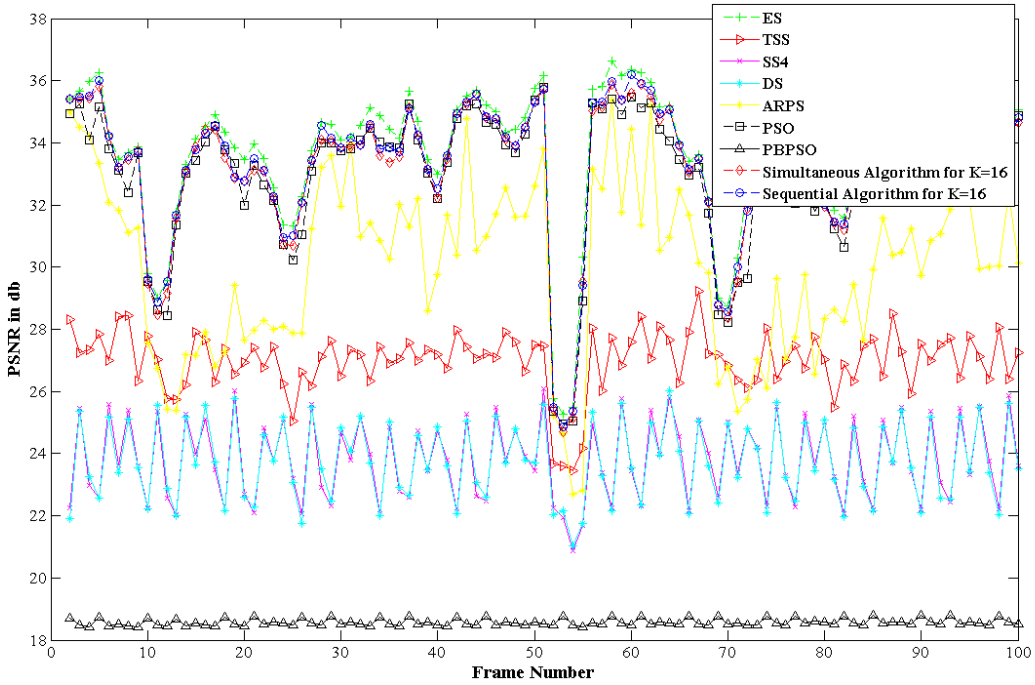


Figure 3.11 Motion estimation quality measured in PSNR for "Container, CIF" sequence

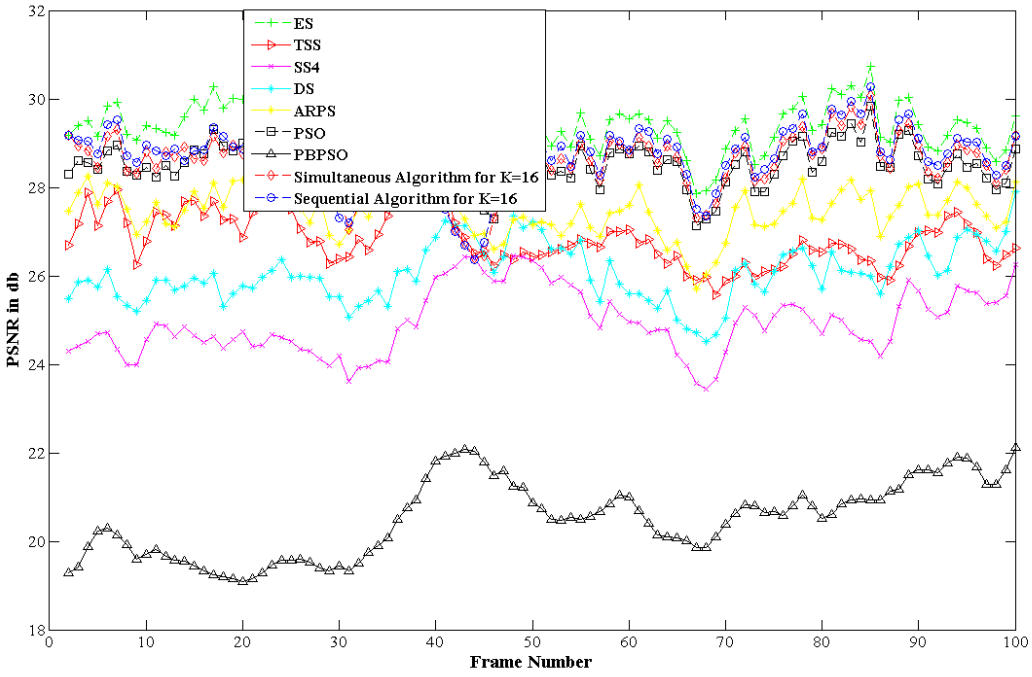


Figure 3.12 Motion estimation quality measured in PSNR for “Racehorses, 480p” sequence

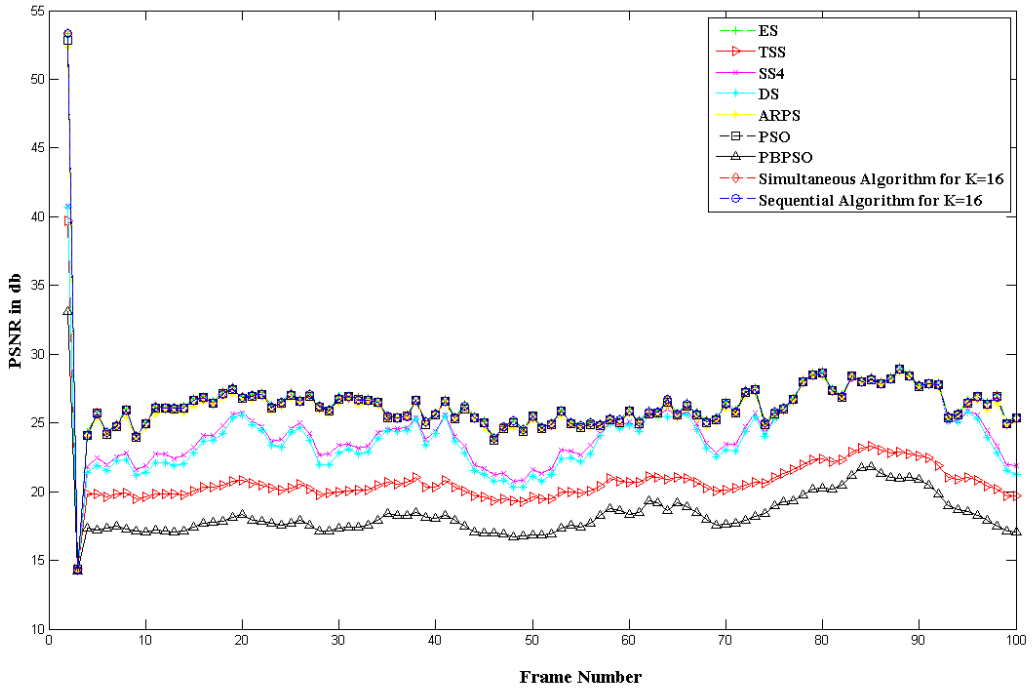


Figure 3.13 Motion estimation quality measured in PSNR for “Parkrun, 720p” sequence

3.5.5 Computational complexity

In block matching motion estimation, the average number of candidate blocks checked for each MB is used as the evaluation criterion of computation complexity. In this paper, the average number of fitness function evaluations for each MB is used as a metric of the computational complexity. This section gives the simulation results of a single core implementation of our proposed algorithms. Table 3.8 provides a comparison of the computational complexity between the proposed sequential and simultaneous algorithms for different number of players. It can be seen from Table 3.8 that the simultaneous algorithm provides a reduction in the computational complexity as compared to the proposed sequential algorithm for the same number of players. The reason behind this is that the sequential algorithm has to go through T_{conv} BR rounds, as shown in Table 3.5, before convergence, whereas the proposed simultaneous algorithm goes through only 2 BR rounds followed by an efficient final MV search. This increases the number of search points of the proposed sequential algorithm as compared to the simultaneous approach. On the other hand, as shown in Table 3.8, increasing the number of players leads to an increase in the computational complexity which is due to the increase in the number of BR rounds needed for convergence for the sequential algorithm and the increase in the computational requirements of the final MV search step in the proposed simultaneous approach.

In Table 3.9, the computational complexity of the proposed algorithms, when 16 players are used, is compared with that of existing techniques. Although the simulation parameters, as shown in section 6.2, are chosen so that the proposed algorithms have a computational complexity equivalent to that of the centralized PSO approach in [52]; nevertheless, it is noticed from the results in Table 3.9 that the proposed distributed approaches provide a reduction in the computational complexity than the PSO algorithm in [52]. The exploitation of time-space

correlation of video sequences through effective particle initialization, agent's fitness calculation history preservation, and the efficient termination strategies used have decreased the number of search points needed. The proposed simultaneous algorithm provides a further reduction in the computational complexity that goes below the PBPSO algorithm in [51] for most of the sequences.

Table 3.8 Comparison of the average number of fitness function evaluations per MB for the proposed sequential and simultaneous algorithms for different values of K

	Sequential Algorithm		**Simultaneous Algorithm**	
Sequence	$K = 4$	$K = 16$	$K = 4$	$K = 16$
Soccer, QCIF, 15fps, p=15	10.99	14.366	9.59	13.67
Bus, QCIF, 15 fps, p=15	10.9	16.055	8.94	12.54
Soccer, CIF, 30fps, p=15	8.89	11.73	7.34	9.57
Bus, CIF, 30 fps, p=15	9.59	14.075	7.9	10.95
Tennis, CIF,30 fps, p=15	8.2	12.95	6.389	9.107
Stefan, CIF, 30fps, p=15	7.89	11.08	7.625	8.687
Foreman, CIF, 30fps, p=15	9.12	11.38	8.82	9.528
Container, CIF, 30fps, p=15	6.29	10.809	7.67	7.268
RaceHorses, 480p, 30 fps, p=15	14.017	16.401	13.29	15.32
Parkrun, 720p, 30 fps, p=31	8.2	13.91	8.202	8.38

Table 3.9 Comparison of the average number of fitness function evaluations per MB for various algorithms based on the first 100 frames of the video sequences

Sequence	**ES**	**TSS**	**4SS**	**DS**	**ARPS**	**PSO [52]**	**PBPSO[51]**	**Simultaneous Algorithm $K = 16$**	**Sequential Algorithm $K = 16$**
Soccer, QCIF,15fps, p=15	961	29.33	18.33	17.66	13.25	16.2	11.024	13.67	14.366
Bus, QCIF,15fps, p=15	961	29.46	19.85	22.51	12.44	19.22	12.482	12.54	16.055
Soccer, CIF,30fps, p=15	961	31.13	20.01	19.89	10.61	13.54	12.24	9.57	11.73
Bus, CIF, 30fps, p=15	961	31.23	24.24	21.39	12.35	17.83	11.92	10.95	14.075
Tennis, CIF,30 fps, p=15	961	30.973	18.893	17.195	9.448	15.839	12.243	9.107	12.95
Stefan, CIF, 30fps, p=15	961	30.753	18.803	18.023	8.819	14.098	11.289	8.687	11.08
Foreman, CIF, 30fps, p=15	961	30.7602	18.456	16.661	8.978	12.925	12.114	9.528	11.38
Container, CIF, 30fps, p=15	961	31.165	21.247	23.0984	9.7977	12.337	12.751	7.268	10.809
RaceHorses, 480p, p=15	961	32.17	30.06	23.35	14.97	16.6	13.31	15.32	16.401
Parkrun, 720p, p=31	3969	40.11	22.97	21.18	9.77	15.924	12.182	8.38	13.91

3.5.6 Parallel performance of the proposed simultaneous algorithm

The parallel version of the proposed simultaneous algorithm is implemented using Matlab PCT. The algorithm is simulated using different Matlab workers, or labs, and the average execution times per frame are recorded in Tables 3.10 and 3.11 for $K = 4$ and $K = 16$ respectively. T1 is the time needed to initialize the swarms of PSO particles for the subblocks, T2 is the total time needed for inter-agent communication, and T3 is the time needed to perform the simultaneous BR rounds and the final MV search step. The parallel performance of our algorithm is evaluated in terms of the speedup factor, parallel efficiency, and granularity.

Table 3.10 Parallel performance of the proposed Simultaneous algorithm using Matlab PCT for $K = 4$

Sequence	Number of Labs	T1 (s)	T2 (s)	T3 (s)	Total Time (s)	Speedup	Efficiency %	Granularity
Soccer QCIF, 15 fps, p=15								
	1	0.0351	0	0.1075	0.1426	1	100	-
	2	0.0236	0.009	0.0671	0.0997	1.4303	71.5145	10.0778
	4	0.015	0.008	0.0367	0.0597	2.3886	59.7152	6.4625
Bus, CIF, 30 fps, p=15								
	1	0.1482	0	0.4034	0.5516	1	100	
	2	0.097	0.0294	0.2398	0.3662	1.5063	75.3140	11.4558
	4	0.0621	0.033	0.1352	0.2303	2.3951	59.8784	5.9788
RaceHorses, 480p, 30 fps, p=15								
	1	0.6465	0	2.4153	3.0618	1	100	
	2	0.3419	0.1089	1.3309	1.7817	1.7185	85.9236	15.3609
	4	0.201	0.1254	0.7907	1.1171	2.7408	68.5212	7.9083
Parkrun, 720p, 50 fps, p=31								
	1	1.3481	0	4.9274	6.2755	1	100	
	2	0.8691	0.3378	2.9044	4.1113	1.5264	76.3201	11.1708
	4	0.5816	0.3948	1.5955	2.5719	2.4400	61.0006	5.5144

Table 3.11 Parallel performance of the proposed Simultaneous algorithm using Matlab PCT for $K = 16$

Sequence	Number of Labs	T1 (s)	T2 (s)	T3 (s)	Total Time (s)	Speedup	Efficiency %	Granularity
Soccer QCIF, 15 fps, p=15								
	1	0.0355	0	0.1053	0.1407	1	100	
	2	0.0204	0.0089	0.0619	0.0912	1.5427	77.1328	9.2509
	4	0.0120	0.0104	0.0322	0.0545	2.5804	64.5111	4.2444
	8	0.0065	0.0127	0.0167	0.0359	3.9204	49.0048	1.8268
	16	0.0035	0.0148	0.0095	0.0278	5.0626	31.6416	0.8784
Bus, CIF, 30 fps, p=15								
	1	0.1648	0	0.5088	0.6736	1	100	
	2	0.1002	0.041	0.2968	0.4380	1.5381	76.9028	9.6817
	4	0.0576	0.0461	0.1656	0.2692	2.5020	62.5511	4.8398
	8	0.0314	0.0511	0.0819	0.1644	4.0973	51.2159	2.2172
	16	0.0210	0.059	0.0512	0.1312	5.1341	32.0880	1.2237
RaceHorses, 480p, 30 fps, p=15								
	1	0.6932	0	2.972	3.6655	1	100	
	2	0.3678	0.142	1.563	2.0727	1.7685	88.4254	13.5963
	4	0.2151	0.168	0.884	1.2667	2.8938	72.3443	6.5399
	8	0.1507	0.213	0.512	0.8757	4.1858	52.3230	3.1113
	16	0.0652	0.251	0.288	0.6042	6.0668	37.9173	1.4072
Parkrun, 720p, 50 fps, p=31								
	1	1.455	0	6.01575	7.47075	1	100	
	2	0.8573	0.337	3.624	4.8183	1.5505	77.5242	13.2977
	4	0.4713	0.4167	1.9695	2.8575	2.6144	65.3601	5.8575
	8	0.313	0.491	1.015	1.8190	4.1071	51.3383	2.7047
	16	0.184	0.526	0.621	1.3310	5.6129	35.0805	1.5304

3.5.6.1 Speedup

To measure the parallel performance of our proposed algorithm, we used the speedup factor, *S(n),*which is defined as:

$$S(n) = \frac{T_s}{T_n}, \tag{3.16}$$

where T_s is the total execution time on a single processor, while T_n is the total execution time on a multicore system of n processors. In Tables 3.10 and 3.11, T_s is taken as the total time when executing the code on one *lab* and T_n is the total time when executing the code on *n labs*. Fig. 3.14 shows a plot of the speedup as function of the number of cores for the four sequences for the two

values of K. In the presented algorithm, for a small number of cores, the amount of communication needed is low since most of the neighbors of a given player will be allocated to the same cores. As shown in the Tables 3.10 and 3.11, the average speedup is 1.6 for two cores. Nevertheless, as the number of cores increases, the speedup also increases but its rate of increase slightly diminishes due to the increase in the communication time resulting from the fact that the neighbors of a given player might reside on different cores.

Note that in the proposed parallel simultaneous ME algorithm, the MBs within the frame are processed sequentially, but the ME process of each MB is parallelized across multiple cores. The motion estimation process of each MB is formulated as a game between K players that are equally distributed across the available processing cores for load balancing. Theoretically, the algorithm is parallelizable across a maximum number of cores equal to the number of players in the game, where each player is assigned to a core or Matlab worker. Therefore, in the proposed algorithm, parallelism is limited to $\mathrm{n} = K$ and using a larger number of cores will be useless.

Comparing the results in Tables 3.10 and 3.11, it is noticed that decreasing the number of players in the game leads to a slight decrease in both the computation and communication times. The reason behind this lies in two folds. First, decreasing the number of players in the game reduces the number of allocated players per core as well as the number of direct neighbors for each player which leads to a decrease in the needed inter-processor communication. Second, using a smaller number of players slightly decreases the needed computations in the final MV search step. The needed computational complexity during the BR rounds is basically the same since its independent of the number of players K as was shown in section E.5. The main limitation in using a smaller value of K is that parallelism is limited to $\mathrm{n} = K$and reducing the number of players would limit the scalability of the algorithm.

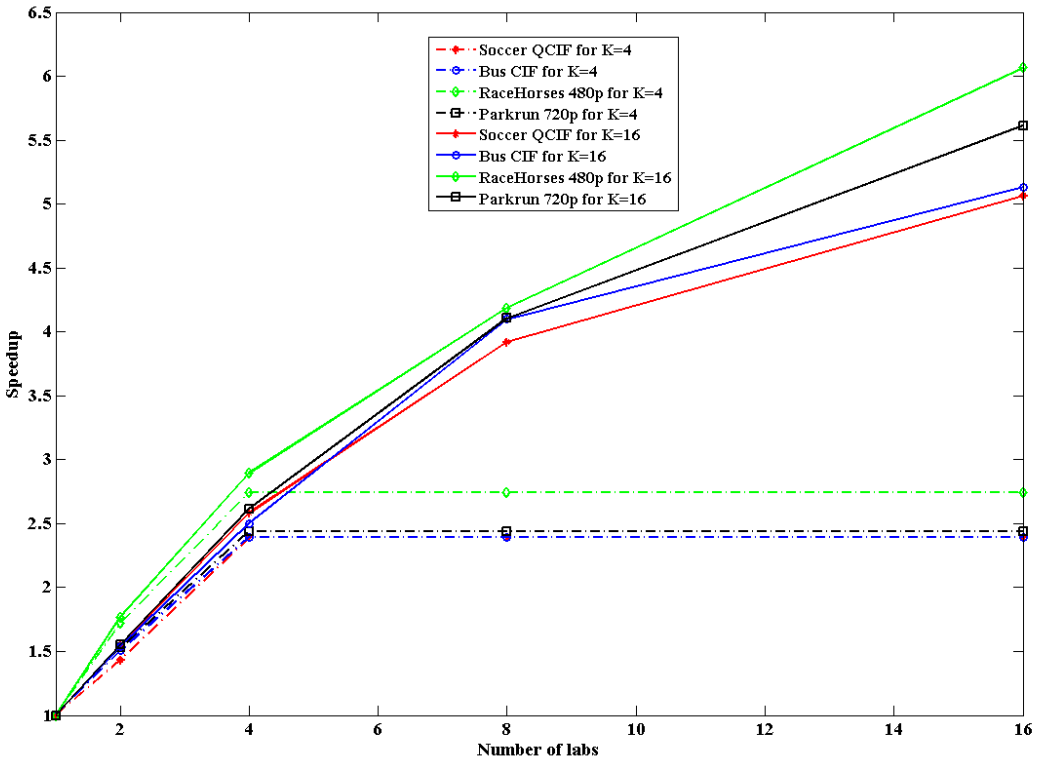


Figure 3.14 Speedup achieved by the parallel implementation of the proposed simultaneous algorithm for different values of K

3.5.6.2 Parallel Efficiency

It is observed that the parallel efficiency decreases with the increase of the number of cores. This is due to the fact that as the number of cores increases, the probability that the neighbors of each player would be allocated to a different core increases. As a result, this leads to an increase in the needed inter-processor communication during the synchronization stage of the algorithm.

3.5.6.3 Granularity

To measure the granularity of our algorithm, the computation and communication times

are taken as:

$$T_{Computation} = T_1 + T_3, \quad (3.17)$$

$$T_{Communication} = T_2. \quad (3.18)$$

As shown in Tables 3.10 and 3.11, the granularity decreases with the increase of number of cores since less number of players would be assigned to each core which results in a decrease in the computation time. Moreover, as the number of cores increases, the amount of communication needed also increases which results in an increase in the communication time. Comparing the values in Tables 3.10 and 3.11, we notice that higher granularity values are recorded for K=4 as compared to the values for K=16 when the same number of cores (for $n \leq 4$) are used. This is due to the decrease in the needed communication needed with the decrease in the number of players, as mentioned before. Overall, in the presented implementation granularity is considered coarse which implies more opportunity for performance increase.

3.6 Summary

This work presents a novel distributed approach to block motion estimation. The optimization problem of BME of a given MB is cast in a game-theoretic setting using a network of autonomous agents. It is shown that by using local communication and applying simple robust state-changing rules such as following natural game-theoretic dynamics, agents can, in a distributed fashion, optimize the global objective function of the whole MB. First, a global objective function that captures the notion of consensus is established. Next, it is shown that local utility functions can be assigned to the players, so that the resulting game is proven to be a potential game. Sequential and simultaneous algorithms based on BRD are proposed to solve the game in a distributed fashion. Theoretical and empirical analysis is provided to prove the convergence of the proposed sequential algorithm. On the other hand, a sub-optimal final MV

search method is proposed in the presented simultaneous scheme to overcome the absence of guaranteed convergence. Despite its lack of guaranteed convergence, the proposed simultaneous algorithm provides a high level of data parallelism between the subblocks. The multi-core implementation of this scheme shows that speedup is indeed obtained. Simulation results show that the proposed algorithms provide an improvement in estimation quality and a decrease in computational complexity as compared to existing techniques.

CHAPTER 4

A DISTRIBUTED PARTICLE SWARM OPTIMIZATION ALGORITHM USING THE STRATEGIES OF DIFFUSION ADAPTATION

Many variants of particle swarm optimization (PSO) [45-52] were used for the problem of ME. Even though these algorithms have very powerful global optimization capabilities, they are, however, highly centralized at the block level and require a central processor to continuously communicate with all the particles in the swarm during the iterative search process. This makes these algorithms very hard to parallelize and thus cannot be accelerated using the available parallel processing technologies. In chapter 3, distributed PSO algorithms were proposed using non-cooperative game theory. The proposed distributed parallel algorithm could achieve parallelism within the MB. In this chapter, we tackle the same problem but from a different perspective. A distributed PSO algorithm is developed to achieve parallelism within the MB using the diffusion adaptation theory in a multi-agent setting.

In multi-agent cooperation problems, different network topologies will influence different manners of cooperation between agents. A centralized system will directly control the operation of each agent with information flow from a single center, while in a distributed system, agents operate separately under certain communication protocols. For distributed multi-agent systems, there are two main challenges need to be addressed to achieve cooperation among a potentially large number of involving agents. First, there is limited information for agents to utilize to achieve the global objective. Moreover, the information of the distributed network topology is unknown to agents.

Diffusion adaptation is an emerging adaptation mechanism that is applied to networks of nodes to solve several types of optimization problems. This mechanism is distributed where nodes are allowed to communicate only with their neighbors. Thus, no centralized processing is needed. In this paper, we study the distributed optimization of block motion estimation using a network of cooperative nodes based on diffusion protocols. The ME problem is formulated as the optimization of a global cost function that is the sum of individual sub-problems. Nodes are equipped with local estimation capabilities based on PSO. They iteratively produce local estimates using PSO to minimize their local cost functions. Diffusion strategies are employed to allow the agents to cooperate and diffuse information in real-time in order to reach the common minimizer of the global cost function. This approach would be highly parallel since it is distributed (non-centralized) and thus suitable for a parallel implementation.

The rest of the chapter is organized as follows. In section 4.1, a brief review of the concepts of multi-agent networks and diffusion adaptation. In section 4.2, the details of the proposed approach are given. Section 4.3 presents the parallel implementation of the proposed diffusion-PSO algorithm using MATLAB PCT. Simulation results are provided and analyzed in section 4.4. Section 4.5 provides a summary of this chapter.

4.1 Background

4.1.1 Multi-agent systems

In many disciplines such as computer science or robotics, the concept of an *agent* is ubiquitous. The birth of the term "agent" has its roots in computer science, whereby an agent is, roughly, defined as an autonomous computer program. The notion of an agent is quite difficult to define. Although numerous papers on the subject of agents and multi-agent systems have been

written, a tremendous number of definitions exist. In essence, an agent is an entity that has the capabilities of an intelligent person or human being. Due to this characteristic, being able to find a unified definition of an agent is quite tough. Although the definition can vary from one discipline to the other, in general, the main characteristics of an agent are its proactive and intelligent ability to sense its environment, interact with it, and take autonomous decisions. In some sense, the role of an agent is to mimic human behavior in a given technical problem whether it be, for example, in computer science, robotics or control systems. Thus, a multi-agent system is a system composed of multiple interacting intelligent agents that can interact, collaborate, and act together in order to solve different problems. For example, multi-agent systems can be used to solve problems in online trading, software engineering, disaster response, military applications, and modeling social structures [83].

The main challenge in designing multi-agent systems is to be able to allow the agents to somehow simulate the way humans act in their environment, interact with one another, cooperatively solve problems or act on behalf of others, solve more and more complex problems by distributing tasks or enhance their problem solving performances by competition. Clearly, the use of agents and multi-agent systems will be one of the landmark technology in many disciplines in years to come, as it will bring extra conceptual power, new methods and techniques, and advanced design approaches. Consequently, this will essentially broaden the spectrum of applications and expand it beyond the computer world into disciplines such as wireless networks or communications theory.

Independent from its application, a general problem that is of strong interest in multi-agent systems, is the distribution of tasks among the different agents. For instance, it is of importance to study how, a number of agents, can autonomously and intelligently allocate

different tasks among each others using cooperative as well as non-cooperative approaches [83]. In a software system, the tasks can represent, for example, threads or programs that need to be executed. In a control system, the tasks can be points in time or space that the agents are required to attend to. For example, in [84, 85], the problem of enabling a number of vehicle-agents to move to randomly generated tasks is studied in a non-cooperative approach, while in [86, 87], the problem of task allocation in a software system is studied using a heuristic coalition formation approach. Additional approaches for agents task allocation in robotics and artificial intelligence are found in [83–92].

4.1.2 Diffusion Adaptation

In recent years, diffusion adaptation strategies have been proposed for the solution of estimation [93, 94] and optimization problems [95] over networks in an adaptive and distributed manner. In [93], authors formulate and study distributed estimation algorithms based on diffusion protocols to implement cooperation among individual adaptive nodes. The individual nodes are equipped with local learning abilities. They derive local estimates for the parameter of interest and share information with their neighbors only, giving rise to peer-to-peer protocols. The resulting algorithm is distributed, cooperative and able to respond in real time to changes in the environment. It improves performance in terms of transient and steady-state mean-square error, as compared with traditional non-cooperative schemes. Closed-form expressions that describe the network performance in terms of mean-square error quantities are derived, presenting a very good match with simulations.

Each node in the network could function as an individual adaptive filter whose aim is to estimate the parameter of interest through local observations. These individual estimates across the

nodes could then be locally fused with their neighboring estimates in the network in order to obtain an estimate that is influenced by the data at the nearby nodes; for instance, by resorting to consensus implementations.

In [96], an iterative diffusion mechanism is developed to optimize a global cost function in a distributed manner over a network of nodes. The cost function is assumed to consist of a collection of individual components, and diffusion strategy allows the nodes to cooperate and diffuse information in real-time. Compared to incremental methods, diffusion methods do not require the use of a cyclic path over the nodes and are more robust to node and link failure.

Adaptive networks are well-suited to perform decentralized information processing and optimization tasks and to model various types of self-organized and complex behavior encountered in nature [97]. Adaptive networks consist of a collection of agents with processing and learning abilities. The agents are linked together through a connection topology, and they cooperate with each other through local interactions to solve distributed optimization, estimation, and inference problems in real-time. The continuous diffusion of information across the network enables agents to adapt their performance in relation to streaming data and network conditions; it also results in improved adaptation and learning performance relative to non-cooperative agents. Diffusion adaptation has been used to model the swarming behavior of flocks of birds [98], honey bees [99], and schools of fish [100] adapting to their environment. The diffusion adaptation algorithm is summarized as follows. Nodes or agents behave as adaptive filter with learning (or estimating capabilities). They exchange estimates with their neighborhood, then combine this information with their intermediate estimates before updating at each time step.

4.2 The Proposed Distributed Block Motion Estimation Algorithm Using PSO and Diffusion Adaptation

The problem addressed in this chapter is how to solve the optimization problem of ME in a distributed way through cooperative processing over a multi-agent network.

4.2.1 Problem Formulation

For motion estimation through a block matching (BM) algorithm, the current frame of an image sequence I^t is divided into non-overlapping blocks of N × N pixels. For each template block in the current frame, the best matched block within a search window (S) of size $(2W + 1) \times (2W + 1)$ in the previous frame $I^{t-1}$is determined, where W is the maximum allowed displacement. Under such perspective, BM can be approached as an optimization problem aiming for finding the best MV within a search space.

The most well-known criterion for BM algorithms is the sum of absolute differences (SAD). Considering a template MB at position (x, y) in the current frame and the candidate MB at position $(x + \hat{u}, y + \hat{v})$ in the previous frame I^{t-1}:

$$SAD(\hat{u}, \hat{v}) = \sum_{j=0}^{N-1} \sum_{i=0}^{N-1} |g_t(x + i, y + j) - g_{t-1}(x + \hat{u} + i, y + \hat{v} + j)|, \quad (4.1)$$

where $g_t(.)$is the gray value of a pixel in the current frame I^t and $g_{t-1}(.)$ is the gray level of a pixel in the previous frame I^{t-1}.

The SAD fitness function used in the BMA can be viewed as a global cost function:

$$J_{glob}(\hat{u}, \hat{v}) = SAD(\hat{u}, \hat{v}). \quad (4.2)$$

Consider a collection of K agents interested in estimating the same parameter vector, w.The vector is the minimizer of the global cost function, $J_{glob}(w)$, which the agents seek to optimize.We want to study the distributed optimization of this global cost function. To do that,$J_{glob}(w)$is assumed to consist of the sum of individual components. We start our development

by associating with each agent k an individual cost (or utility) function, $J_k(w)$, such that:

$$J_{glob}(u,v) = \sum_{k=1}^{K} J_k(u,v). \quad (4.3)$$

In this way, the global cost function is divided into a number of local cost functions each associated to an agent. In order to find such decomposition, we will uniformly divide the MB, of dimension NxN, into K equal subblocks of dimensions LxL. Then, a network of K agents is employed such that agent k would minimize the local cost function given by:

$$J_k(\hat{u},\hat{v}) = \sum_{j=0}^{L-1}\sum_{i=0}^{L-1}|g_t(x_k+i, y_k+j) - g_{t-1}(x_k+\hat{u}+i, y_k+\hat{v}+j)|, \quad (4.4)$$

where$(x_k, y_k)$is the position of the subblock.

The MB decomposition which was used in Chapter 3, and shown Fig.3.1, is again used here. The MB, of dimension 16x16 (that is $N = 16$) is divided along the rows and columns into 4x4 equal subblocks. In this case $K = 16$, and each subblock is of dimension 4x4, that is L is 4.

By finding the minimizer of the local cost function, each agent is searching for the optimal MV of each subblock. Therefore, the minimizers of the local cost functions may not coincide and each subblock would converge to a different MV. Our target however, is to find the optimal MV for the entire MB. Therefore, all agents must converge to a common minimum which is the minimizer of the global cost function.

To do this, cooperation between the agents is essential. Diffusion adaptation strategies are employed to allow the agents to share information locally with their neighborhood. By cooperating with their neighbors, and by having these neighbors cooperate with their neighbors, procedures can be devised that would enable all agents in the network to converge towards the global optimum through local interactions.

In diffusion adaptation, neighboring nodes can share information with each other as permitted by the network topology. In this algorithm, we use the network topology which was

proposed in Section 3.2.1 and shown in Fig.3.2. Fig. 3.2 shows a given MB decomposed into 16 sub-blocks with one agent, or node, used for each sub-block. The node of each sub-block is connected with the nodes of its 8 neighboring sub-blocks. Such a topology graph is chosen because in motion estimation there is a high level of spatial correlation. Therefore, we expect that allowing each node to share information with its 8 neighbors would improve the global performance level.

To motivate the distributed diffusion-based approach, we start by introducing a set of nonnegative coefficients $\{c_{kl}\}$ that satisfy two conditions:

for $k = 1,2, \dots, K$:

$$c_{kl} \geq 0, \sum_{l=1}^{K} c_{kl} = 1, \ and \ c_{kl} = 0 \ if \ l \notin \mathcal{N}_k, \quad (4.5)$$

where $\mathcal{N}_k$ denotes the neighborhood of node k.

Nodes or agents then use these coefficients $\{c_{kl}\}$ to fuse estimates from their neighbors at each iteration.

Agents are equipped with local estimation capabilities. They should be able to iteratively estimate the minimizers of their local cost functions. Since the local, as well as, the global cost function, have a lot of local minima, they are classified to be non-convex. Therefore, agents should use a global optimization technique. PSO is chosen as the global optimization algorithm used by the agents due to its profound intelligence and simple algorithm structure.

4.2.2 Proposed Diffusion PSO Block Motion Estimation Algorithm

The proposed diffusion block motion estimation algorithm is implemented using PSO as follows:

An MB is divided into Ksubblocks as shown in Fig. 3.1. Then, a swarm of M particles is defined for each subblock. PSO iterations are performed in each subblock in order to find the

minimizer of the local cost function of the subblock as defined in (4.4). After the completion of an adaptation step through a modified PSO iteration by all subblocks, a diffusion step is performed. The proposed approach follows the Combine-then-Adapt (CTA) diffusion algorithm, first proposed and extended in [93, 94, 101-105] for the solution of distributed mean-square-error, least-squares, and state-space estimation problems over networks. The algorithm consists of two steps: a diffusion (combination) step followed by an adaptation (processing) step. The proposed diffusion PSO model is shown in Fig. 4.1. The details of the proposed algorithm are as follows:

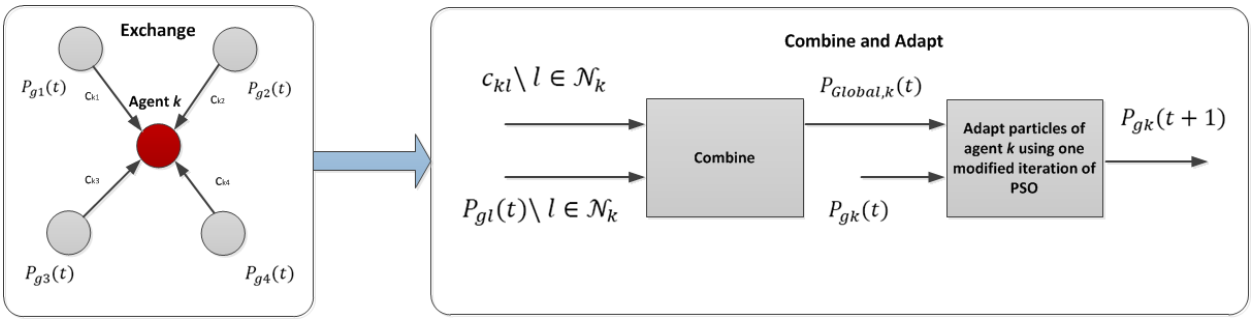


Figure 4.1 Proposed PSO Diffusion Adaptation Model

4.2.2.1 Diffusion Step

In this diffusion step, each agent will receive updated information from its neighboring agents about their current estimates. Using the connectivity graph shown in Fig. 3.2, each subblock k will take a weighted average of the Pg positions found so far by its neighbors denoted by $P_{Global,k}$ which is given by:

$$P_{Global,k} = \sum_{l=1}^{K} c_{kl} * P_{gl}, \qquad (4.6)$$

where the coefficients c_{kl} satisfy (4.5).

4.2.2.2 Adaptation Step

The adaptation step used in the proposed algorithm consists of one PSO iteration performed by all the agents in the MB. The basic PSO iteration, however, is modified to take into consideration the information received from the neighboring agents. This is done by introducing the following modifications into the PSO process:

4.2.2.2.1 Modified PSO Velocity Update Equation

The velocity update equation of PSO in (2.4) is modified to include the contribution of the neighboring subblocks. In addition to the best position found so far by the particle and the best position found so far by all the swarm of the subblock, a third attracting element is introduced which is $P_{Global,k}$ which denotes the information received from the neighboring subblocks. $P_{Global,k}$ is embedded in the velocity update equation of particle i of subblock k as follows:

$$V_{ik}(t+1) = wV_{ik}(t) + c_1 r_1[P_{ik}(t) - X_{ik}(t)] + c_2 r_2\left[P_{gk}(t) - X_{ik}(t)\right] + c_3 r_3\left[P_{Global,k}(t) - X_{ik}(t)\right] \quad (4.7)$$

In this way, a particle in subblock k will not only be attracted towards the minimizer of the local cost function of its subblock but also towards the minimizers of the cost functions of its surrounding subblocks. Following this approach, we aim to drive PSO to find the motion vector that would minimize the local SAD of the subblock while being very close to the motion vectors of the surrounding MBs.

4.2.2.2.2 Modified Fitness Function

The fitness function used in the PSO iterations to evaluate any found motion vector $(\hat{u}, \hat{v})$ is also modified to include not only the SAD of the subblock but also a regularization term that measures the deviation of the motion vector from the motion vectors found so far by the neighboring subblocks. This is done as follows:

$$F(\hat{u},\hat{v})_k = J_k(\hat{u},\hat{v}) + \alpha(t) * \left|(\hat{u},\hat{v}) - P_{Global,k}\right|, \quad (4.8)$$

where $F(\hat{u},\hat{v})_k$ is the fitness of the motion vector $(\hat{u},\hat{v})$ for subblock k, and $\alpha(t)$ is the regularization factor.

The regularization factor $\alpha(t)$ is a used to adaptively varied in each PSO iteration t as follows:

$$\alpha(t) = \frac{t}{N_t}, \quad (4.9)$$

$\alpha(t)$ is linearly increasing with the number of iterations. When t is small, i.e. in the beginning of the PSO iterative process, the fitness value of a given search point is the real cost function of the subblock. As we go through the PSO process, i.e. t increases, then the effect of the regularization term increases and more weight will be given to the deviation from the neighboring subblocks. Thus, we ensure that the motion vector estimated by a given agent is aligned with the average estimates of its neighbors. The objective of using this approach is to drive all the PSO swarms of the subblocks eventually to reach consensus on a common motion vector for all the subblocks which is the optimum motion vector of the whole MB.

The pseudo code of the proposed algorithm is shown in Table 4.1.

Table 4.1 Pseudo code of the proposed diffusion-PSO algorithm

For each MB in the frame **do**
Initialize a set of M particles
For each agent or subblock k=1,…,K **do**
Initialize the fitness history array entries to zeros
Initialize particle velocities to zeros
End for
PSO iterations:
Repeat
Update the regularization factor $\alpha(t)$ using (4.9)
Diffusion Step:
All Agents broadcast their actions to their neighbors.
All agents receive information from their neighbors about their actions
Adaptation step:
For each agent or subblock k=1,…,K **do**
For each particle i=1,…,M **do**
Check its flag in agent k history array
If the flag is 0 then
Calculate utility function of the agent
Update the particle's best position P_i and the global best position P_g of the agent
Save the value of the fitness value in the history array
Set flag to 1
Else
Retrieve the value of the fitness function from the history array
Update P_i
End if
Adaptively change v_{max}
Update the velocity using (4.7)
Update the position
End for
End for
Until stopping conditions of the PSO process are met
If *convergence* is not reached then
Final MV search step
End If
End for

4.2.3 Computational complexity analysis

As was shown in section 3.3.1.9 in chapter 3, the computing complexity of calculating the SAD is $O(N^2)$, where N is the number rows or columns in NxN square MB. In the proposed diffusion-based PSO algorithm, each MB is divided into K subblocks as shown in Fig. 4.2. The cost function of agent k is the SAD of the subblock of dimension LxL as given in (4.4). Therefore, each SAD computation performed by an agent is $O(L^2)$. Since, as illustrated in Fig. 4.2,

$N^2/L^2 = K$, then we can say that each cost function computed by an agent corresponds to $1/K$ MB fitness function evaluation.In the proposed algorithms, we have K agents within a given MB. In the proposed process, the M particles of each agent will perform a maximum of N_t PSO iterations. Therefore, the maximum total computational complexity of the proposed algorithm, in terms of fitness function evaluations per MB, is given by:

$$Computational\ Complexity = K * \frac{N_t * M}{K} = N_t * M. \quad (4.10)$$

4.3 Parallel Implementation

In this section, a multicore implementation of the proposed diffusion-PSO algorithm is proposed using the MATLAB® Parallel Computing Toolbox™ (PCT). From the pseudo code of the proposed diffusion PSO algorithm given in Table 4.1, we notice that the proposed ME algorithm for each MB in the frame is made up of the following steps: Initialization step to initialize the PSO swarms for the agents followed by the diffusion PSO iterations. The initialization step for each agent is completely independent from that of the other agents. Therefore, it can be parallelized. Moreover, each iteration of the proposed diffusion PSO process is made up of two steps: a diffusion step followed by an adaptation step. Both of these steps are inherently parallel between the agents. Therefore, a simultaneous execution of the diffusion step can be performed by all the agents followed by a simultaneous execution of the adaptation step.

The details of the proposed parallel implementation are shown in Table 4.2.

Table 4.2 Pseudo code of the parallel implementation of the proposed Diffusion-PSO algorithm using MATLAB

For each MB in the frame **do**
Initialize a set of M particles
spmd
For k=***drange*** (1,…,K) **do**
Initialize the fitness history array entries to zeros
Initialize particle velocities to zeros
End ***for drange***
PSO iterations:
Repeat
Update the regularization factor $\alpha(t)$ using (4.9)
Diffusion Step:
All agents exchange information from their neighbors about their actions using ***labSendReceive***
For k=***drange*** (1,…,K) **do**
Adaptation step:
For each particle i=1,…,M **do**
Check its flag in agent k history array
If the flag is 0 then
Calculate utility function of the agent
Update the particle's best position P_i and the global best position P_i of the agent
Save the value of the fitness value in the history array
Set flag to 1
Else
Retrieve the value of the fitness function from the history array
Update P_i
End if
Adaptively change v_{max}
Update the velocity using (4.7)
Update the position
End for
End ***for drange***
Until stopping conditions of the PSO process are met
If *convergence* is not reached then
Final MV search step
End If
End for

4.4 Simulation results

4.4.1 Experimental setup

Several test video sequences of various formats and various motion intensity, (QCIF: 176x144), LD (CIF: 352x288), SD (480p: 832x480), and HD (720p: 1280x720), have been used to test the performance of our proposed algorithm and compare it to existing techniques. In our simulations, every frame is divided into MBs of size 16 * 16 pixels. The search parameter W which defines the search area is chosen to be 15 except for the HD (Parkrun) video sequence where

W was chosen to be 31.

For the PSO algorithm, the size of the particle population was chosen to be M=10 and the maximum number of iterations is N_t = 3. The stopping conditions used in [52] are adopted in the simulations. The maximum allowed velocity V_{max} = 15 for a search range W of ±15 and V_{max} = 31 for a search range W of ±31.

Results are presented with two distinct criteria: objective motion estimation quality and computational complexity.

4.4.2 Motion estimation quality

Objective motion estimation quality is measured interms of Peak Signal to Noise Ratio (PSNR) values averaged over the first 100 frames of each test video sequence.

Table 4.3 gives the average PSNR results for the ES algorithm and several traditional fast searching techniques, like three step search (TSS) [16], four step search (4SS) [18], diamond search (DS) [19], and adaptive root pattern search (ARPS)[22]. PSNR results are also given for the recently proposed PSO-based ME algorithms given in [51, 52]. Simulation results show that the proposed algorithm provides an improvement in motion estimation quality as compared to the other techniques as shown in Fig. 4.2, Fig. 4.3, Fig. 4.4, and Fig. 4.5.

Table 4.3 Motion estimation quality in terms of PSNR of the proposed Diffusion-PSO algorithm as compared to existing techniques

Algorithm	**Sequence**			
	Soccer QCIF, W=15	***Bus CIF, W=15***	***RaceHorses 480p, W=15***	***Parkrun 720p, W=31***
ES	25.01	25.61	29.34	25.61
TSS	24.02	22.37	26.81	20.44
4SS	22.11	19.79	24.891	23.66
DS	23.27	20.34	26.02	23.31
ARPS	23.77	21.79	27.45	25.33
PSO[8]	24.33	24.9	28.07	24.4
PBPSO[9]	19.12	17.45	20.43	19.1
Diffusion-PSO	24.38	25.39	28.95	25.54

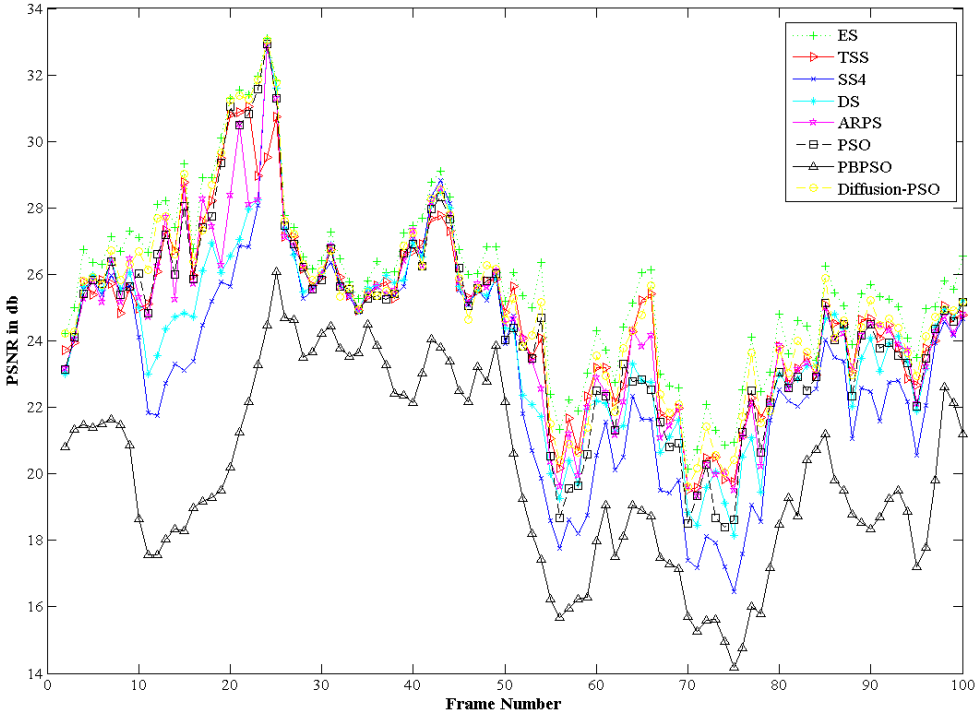


Figure 4.2 Motion estimation accuracy measured in PSNR for "Soccer QCIF" sequence.

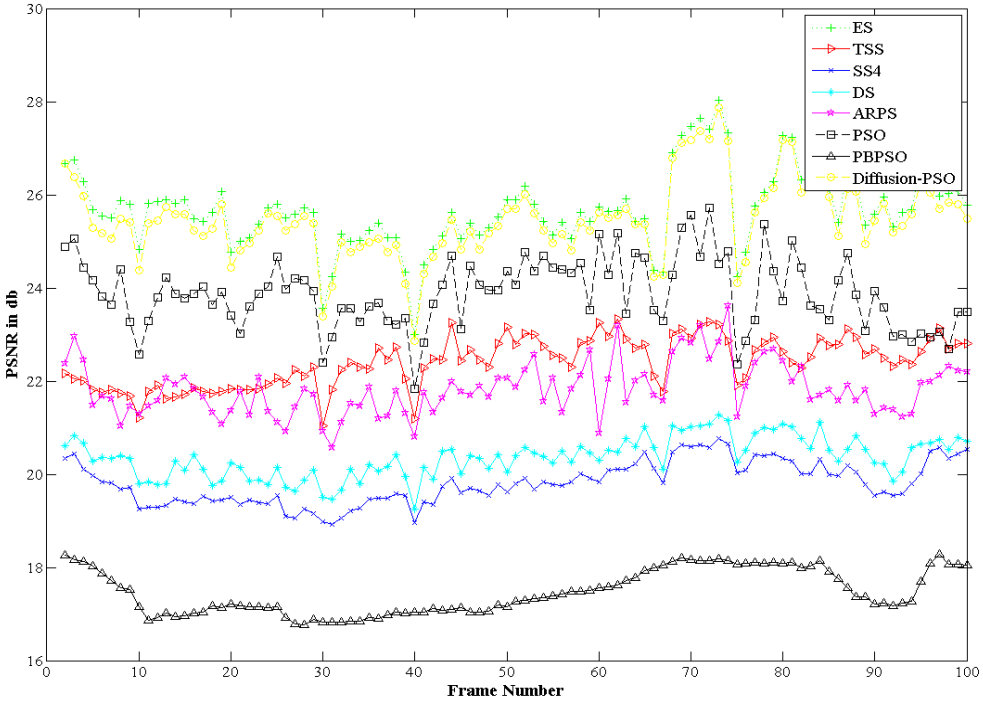


Figure 4.3 Motion estimation accuracy measured in PSNR for "Bus CIF" sequence.

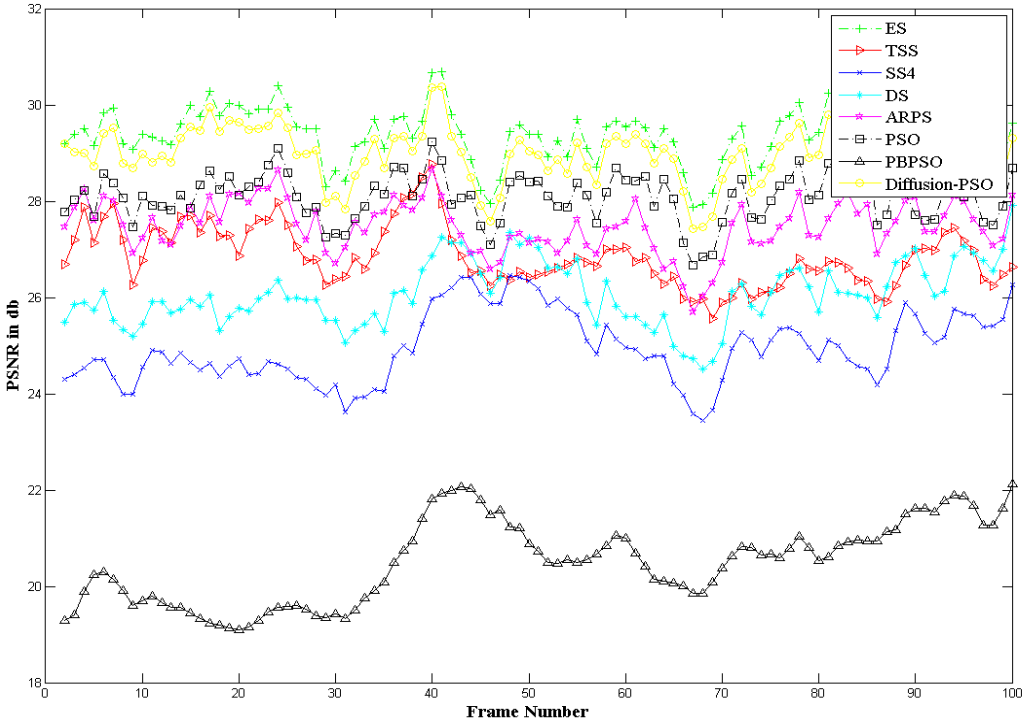


Figure 4.4 Motion estimation accuracy measured in PSNR for "RaceHorses 480p" sequence.

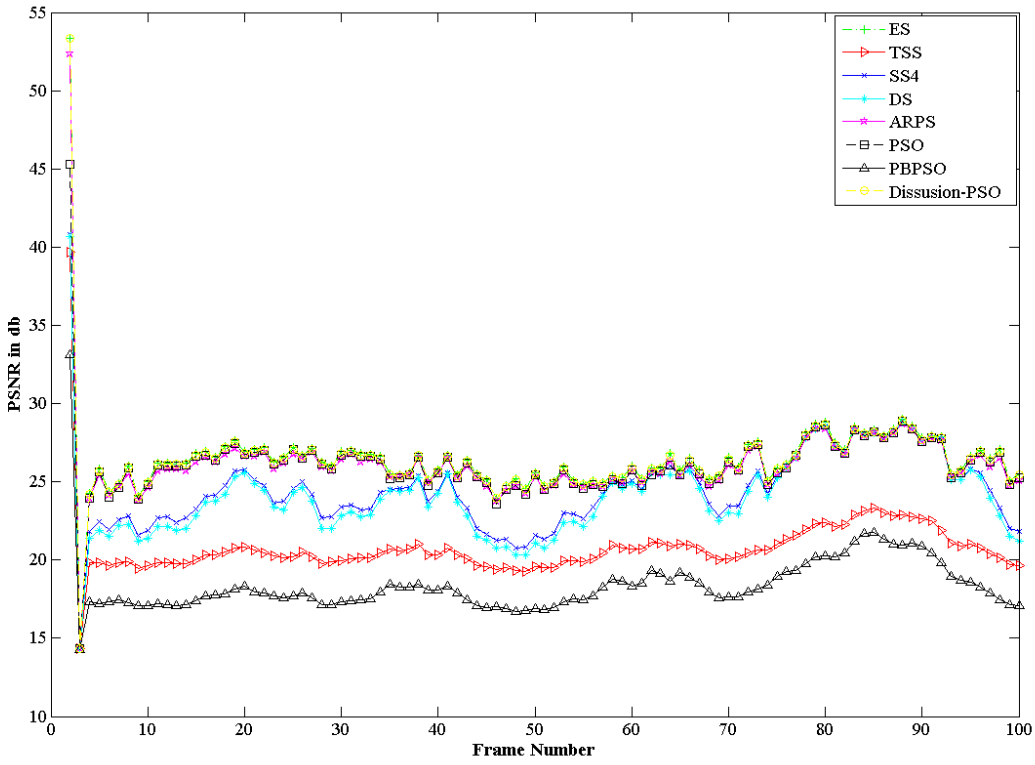


Figure 4.5 Motion estimation accuracy measured in PSNR for "Parkrun" sequence.

4.4.3 Computational complexity

The average number of fitness function evaluations for each MB is used as a metric of the computational complexity. As shown in Table 4.4, the proposed approach provides a significant decrease in the computational complexity as compared to existing techniques.

Table 4.4 Comparison of the average number of fitness function evaluations per block of the proposed Diffusion-PSO algorithm based on the first 100 frames of the video sequences

Algorithm	**Sequence**			
	Soccer QCIF, W=15	***Bus CIF, W=15***	***RaceHorses 480p, W=15***	***Parkrun 720p, W=31***
ES	961	961	961	3969
TSS	29.33	31.23	32.17	40.11
4SS	18.33	24.24	30.06	22.97
DS	17.66	21.39	23.35	21.18
ARPS	13.25	12.35	14.97	9.77
PSO[52]	14.53	15.83	16.01	15.024
PBPSO[51]	11.024	11.92	13.31	12.182
Diffusion-PSO	12.94	10.03	11.61	7.98

4.4.4 Parallel Performance

The parallel version of the proposed diffusion-PSO algorithm is implemented using Matlab PCT. The algorithm is simulated using different Matlab workers, or labs, and the average execution times per frame are recorded in Table 4.5. T1 is the time needed to initialize the swarms of PSO particles for the subblocks, T2 is the total time needed for inter-agent communication, and T3 is the time needed to perform the modified PSO iterations and the final MV search step. The parallel performance of our algorithm is evaluated interms of the speedup factor, parallel efficiency, and granularity.

Table 4.5 Parallel performance of the proposed diffusion-PSO algorithm using MATLAB PCT

Sequence	Number of Labs	T1 (s)	T2 (s)	T3 (s)	Total Time (s)	Speedup	Efficiency %	Granularity
Soccer QCIF, 15 fps, p=15								
	1	0.0573	0	0.0951	0.1524	1	100	
	2	0.0289	0.0198	0.0489	0.0976	1.5606	78.0293	3.9308
	4	0.0174	0.02486	0.0283	0.0706	2.1593	53.9824	1.8383
	8	0.0093	0.03146	0.0154	0.0562	2.7130	33.9120	0.7851
	16	0.0062	0.0374	0.0089	0.0525	2.9010	18.1312	0.4043
Bus, CIF, 30 fps, p=15								
	1	0.2274	0	0.4505	0.6779	1	100	
	2	0.1226	0.08118	0.2595	0.4633	1.4634	73.1700	4.7065
	4	0.0701	0.09878	0.1471	0.3160	2.1452	53.6312	2.1991
	8	0.0413	0.11462	0.0819	0.2378	2.8512	35.6395	1.0744
	16	0.0287	0.1342	0.0517	0.2146	3.1597	19.7483	0.5987
RaceHorses, 480p, 30 fps, p=15								
	1	0.9970	0	2.101	3.0979	1	100	
	2	0.5278	0.3124	1.204	2.0441	1.5156	75.7776	5.5432
	4	0.3296	0.36652	0.805	1.5010	2.0639	51.5973	3.0953
	8	0.1950	0.52096	0.471	1.1874	2.6090	32.6129	1.2792
	16	0.1289	0.5742	0.266	0.9689	3.1974	19.9840	0.6874
Parkrun, 720p, 50 fps, p=31								
	1	2.021	0	5.414	7.4352	1	100	
	2	1.140	0.79104	3.329	5.2604	1.4134	70.6718	5.6500
	4	0.625	0.91608	1.798	3.3393	2.2266	55.6649	2.6452
	8	0.363	1.2168	0.981	2.5602	2.9042	36.3023	1.1040
	16	0.2422	1.3944	0.6127	2.2493	3.3056	20.6602	0.6131

Fig. 4.6 shows a plot of the speedup as function of the number of cores for the four sequences. As shown in the Table 4.5, the average speedup is 1.47 for two cores. Nevertheless, as the number of cores increases, the rate of increase in speedup becomes very low. In fact, the average maximum speedup reached is 3.14 for 16 labs. The reason behind this is the amount of inter-lab communication needed by this algorithm. In the proposed scheme, agents should diffuse information about their estimates before each iteration of PSO. Since the number of PSO iterations used in our simulations is three ($N_t = 3$), this means that three inter-agent communication stages are

needed. This results in diminishing speedup gains with increased number of cores.

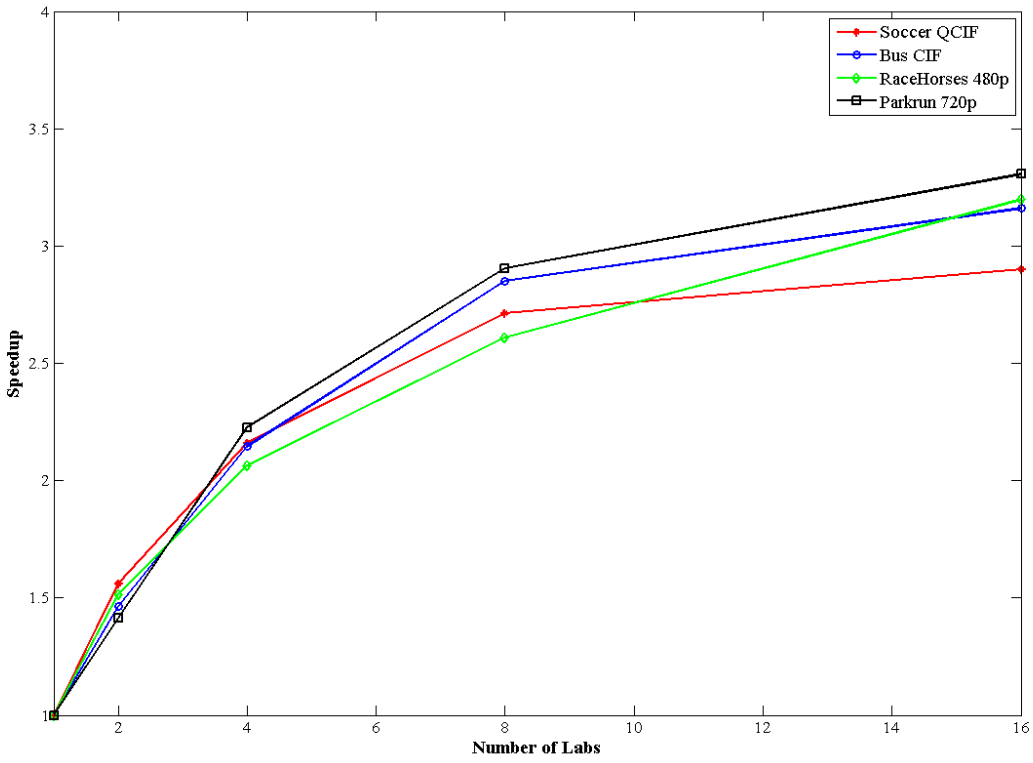


Figure 4.6 Speedup achieved by the parallel implementation of the proposed diffusion-PSO algorithm

4.4.5 Comparison with the proposed simultaneous game-theoretic algorithm

The proposed diffusion-PSO algorithm proposed in this chapter is highly correlated with the proposed simultaneous-BR game-theoretic ME algorithm which was proposed in Chapter 3. In this section, we will highlight their algorithmic difference and provide a comparison in their estimation and parallel performance.

In terms of algorithmic setup, as was mentioned in section 3.5.2, the simultaneous algorithm performs 2 BR rounds ($T = 2$), where in each round, all players or agents simultaneously perform a set of PSO iterations ($N_t = 5$). One communication stage is needed between the BR rounds for players' synchronization. The PSO swarm of each subblock is made up

of three particles (*M=3*). The resulting maximum computational complexity, as was mentioned in section 3.5.2, was found to be 30 ($T * M * N_t = 30$) fitness function evaluations per MB. On the other hand, in the proposed diffusion-PSO algorithm, agents perform simultaneously a total of three PSO iterations with inter-agent communication performed before each iteration. Each agent is equipped with 10 PSO particles which results in a maximum total computational complexity of 30 fitness function evaluations per MB, as was shown in section 4.2.3. Therefore, we notice that both algorithms have equivalent computational requirements but the amount of inter-lab communication needed for the diffusion-PSO algorithm is higher.

The differences in the algorithmic setup of both schemes are translated in their simulation results. In terms of estimation accuracy, comparing the results shown in Tables 3.7 and 4.3, we notice that the diffusion-PSO algorithm provides a slight improvement in the estimation accuracy as compared to the simultaneous game-theoretic scheme. In terms of computational complexity, comparing the results shown in Tables 3.6 and 4.4, we notice the presented diffusion-PSO scheme provides a reduction in the needed fitness function evaluations as compared to the simultaneous algorithm. On the other hand, comparing the parallel performance of the two algorithms shown in Tables 3.11 and 4.5, we notice that the speedup achieved by the simultaneous algorithm is higher than that provided by the proposed diffusion-PSO scheme.

Therefore, we deduce that the two proposed parallel algorithms provide a tradeoff between estimation and parallel performances.

4.5 Summary

A novel approach for BM estimation that achieves parallelism within the MB is presented. The problem is formulated in a distributed multi-agent system where only local communication is

allowed. A diffusion-based PSO process is proposed to drive the agents, in a distributed manner, towards consensus. A novel velocity update equation for PSO is proposed to serve as an adaptation step. A novel PSO fitness function that includes a regularization term is also proposed. Simulation results show that the proposed scheme provides high estimation accuracy with low computational requirements. The multi-core implementation of the proposed algorithm using Matlab PCT shows a speedup of 1.47 on two labs and 3.14 on 16 labs. The limited speedup is due to the multiple inter-agent communication stages needed by the proposed diffusion scheme.

CHAPTER 5

A NOVEL HYBRID DYNAMIC PARTICLE SWARM OPTIMIZATION ALGORITHM FOR MOTION ESTIMATION IN HIGH RESOLUTION VIDEO

Various ME algorithms based on PSO have been proposed in [45-52]. These algorithms provide performance enhancement to the ES algorithm as well as some existing fast searching techniques. The results presented in [45-52], however, are given only for low-definition (LD) video. The available PSO algorithms, when applied on high definition (HD) video, are found to yield a quality worse than that obtained for low definition (LD) video. The reason behind this is that PSO has a major drawback which is that the swarm may prematurely converge. The fast rate of information flow between PSO particles leads to the creation of similar particles. This results in a loss of diversity that increases the possibility of being trapped in local minima where all particles converge to the same point. This problem is not apparent in LD video, but becomes more fundamental in HD video. This is because, as the resolution of the video increases, the number of local minima falls increases because there is a lot of similar information among neighboring pixels (and blocks).The increase in the number of local minima enhances the problem of premature stagnation in the basic PSO algorithm.

In this chapter, a dynamic hybrid evolutionary motion estimation algorithm is proposed. It combines two heuristic optimization techniques: PSO and the Genetic Algorithm (GA). Algorithms based on the genetic algorithm (GA) have been proposed in [106-40, 41]. GA is a stochastic search procedure based on the mechanics of natural selection, genetics and evolution [107]. Since this type of algorithm simultaneously evaluates many points in the search space, it is more likely to find the global solution of the ME problem. Nevertheless,PSO has many advantages

compared to GA. First, it has memory, so knowledge of good solutions is retained by all the particles; whereas in GA, previous knowledge of the problem is discarded once the population changes. Moreover, it allows constructive cooperation between particles where particles in the swarm share information among themselves. In the proposed algorithm, the merits of the GA algorithm are integrated into PSO in order to alleviate its premature convergence and stagnation in HD video. GA operators like selection, breeding, and mutation are applied on PSO particles in an innovative manner in order to increase the diversity of the population. A novel population initialization scheme is proposed that exploits space-time correlation in video sequences in order to improve the convergence rate of the algorithm. The presented algorithm is also dynamic since the maximum allowed velocity of the particles is adaptively varied during the PSO iterative process.

The rest of the chapter is organized as follows. Section 5.1 provides a review of the GA algorithm. Section 5.2 highlights the behavior of existing ME algorithms using PSO when applied to HD video. Section 5.3 presents the proposed hybrid motion estimation scheme. Simulation results are given and analyzed in section 5.4. Finally, section 5.5 summarizes this chapter.

5.1 Basic Concepts of the GA

In GA, a candidate solution for a specific problem is called an individual or a chromosome and consists of a linear list of genes. Each individual represents a point in the search space, and hence a possible solution to the problem. A population consists of a finite number of individuals. Each individual is decided by an evaluating mechanism to obtain its fitness value. Based on this fitness value and undergoing genetic operators, a new population is generated iteratively with each successive population referred to as a generation. The GAs use three basic operators (reproduction,

crossover, and mutation) to manipulate the genetic composition of a population. Reproduction is a process by which the most highly rated individuals in the current generation are reproduced in the new generation. The crossover operator produces two offsprings (new candidate solutions) by recombining the information from two parents. There are two processing steps in this operation. In the first step, a given number of crossing sites are selected uniformly, along with the parent individual at random. In the second step, two new individuals are formed by exchanging alternate pairs of selection between the selected sites. Mutation is a random alteration of some gene values in an individual. The allele of each gene is a candidate for mutation, and its function is determined by the mutation probability. Many efforts on the enhancement of traditional Gas have been proposed [108]. Among them, one category focuses on modifying the structure of the population or the role an individual plays in it [109]–[112], such as distributed GA [110], cellular GA [111], and symbiotic GA [112]. Another category aims to modify the basic operations, such as crossover or mutation, of traditional GAs [113]–[115].

5.2 Motion Estimation in HD Video: Stagnation of PSO particles

It is found that increasing the video resolution can directly affect the accuracy of ME. High resolution videos tend to present very similar neighboring pixels (much more than low resolution ones) and this fact contributes to increase the occurrence of local minima falls [116]. ME algorithms can be affected by this characteristic, generating different results, for the same video, in different resolutions.

The ES motion estimation algorithm as well as the PSO-based ME algorithm proposed in [52] have been applied to four HD 720p video sequences which are: Shields, Parkrun, Stockholm, and Mobcal downloaded from [68]. These sequences have been resized to two lower resolutions:

EDTV 480p (854x480) and LDTV 240p (427x240). The block size used is 16 and the search area p is decreased proportionally with the resolution, where a value of p=42 is taken for the 720p resolution, p=24 is taken for the 480p resolution, and p=12 for the 240p resolution. Fig. 5.1 shows the average peak signal to noise ratio (PSNR) for the four sequences, based on the first 100 frames of each sequence, in the three different resolutions. As can be seen from Fig. 5.1, the performance of the PSO ME algorithm in [52] is very close to that of the ES algorithm at low resolution. The performance gap, however, starts to increase with the increase in the video resolution. The reason behind this is that, as the video resolution increases, the number of local minima falls increases because there is a lot of similar information among neighboring pixels (and blocks).The increase in the number of local minima enhances the problem of premature stagnation in the basic PSO algorithm. Depending on the problem, when searching through the space of solutions, optimizers can stagnate — they cannot find a better solution within a specified amount of time. Some of the reasons stagnation occurs are because the algorithm has no means to escape a local minimum it is currently trapped in – thus leading to premature convergence, or because it moves along a large plateau, or maybe it jumps on an equally sized densely spiked region [117]. Indeed, as the video resolution increases, the search region becomes more densely spiked due to the increase in the number of local minima falls. This is shown in Fig. 5.2 which gives 3D maps of the fitness function, taken as the mean squared error (MSE) between an original block and a candidate block, over the entire search area. Fig. 5.2 shows the 3D maps of the fitness function for three different resolutions of the Parkrun sequence. Each map represents the same region of the frame, with a different number of pixels. Figures 5.2(a)–2(c) represents the MSE maps for the resolutions 240p, 480p, and 720p respectively. The images represent the MSE value for 16×16 blocks, where valleys represent lower MSE values, and peaks represent higher MSE values.

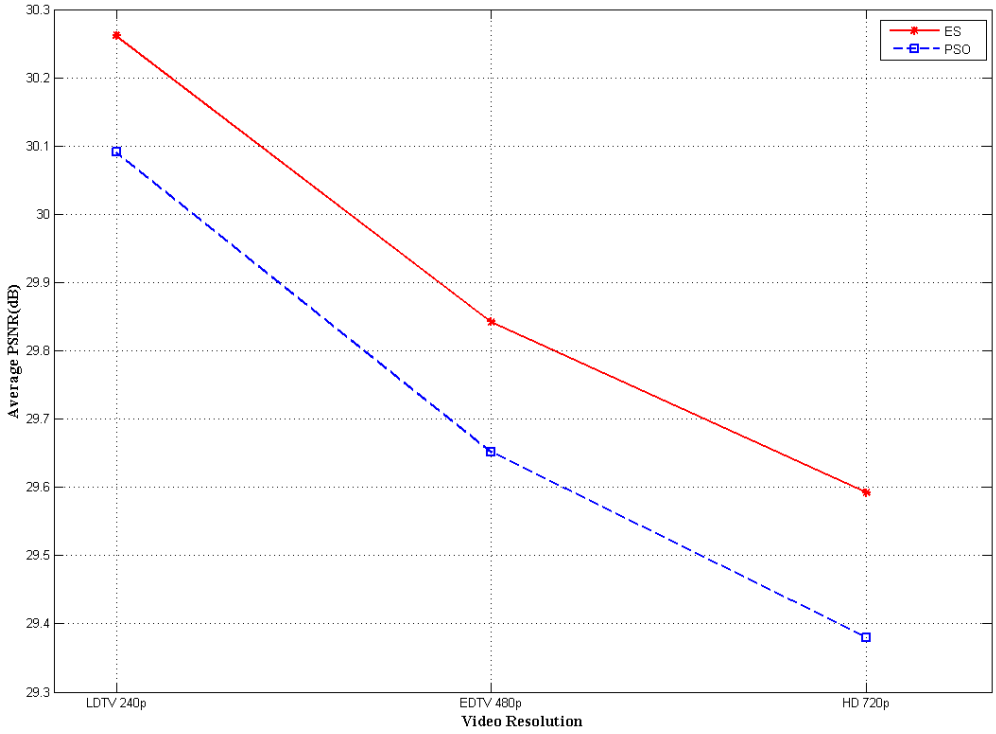
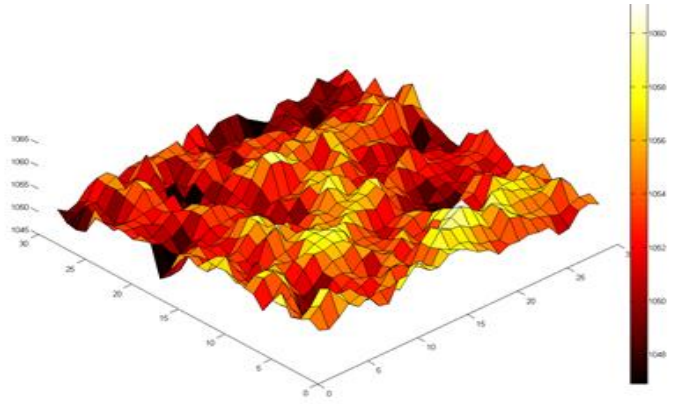
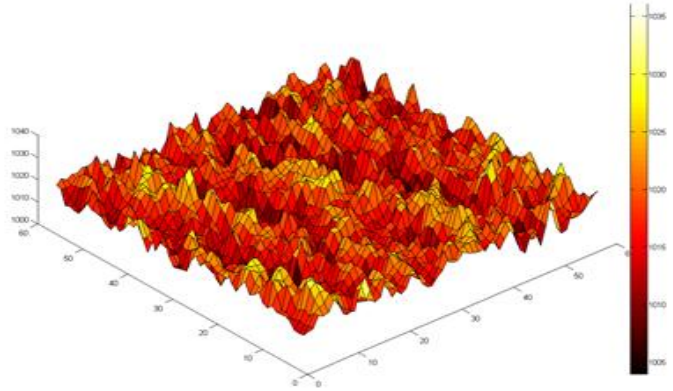


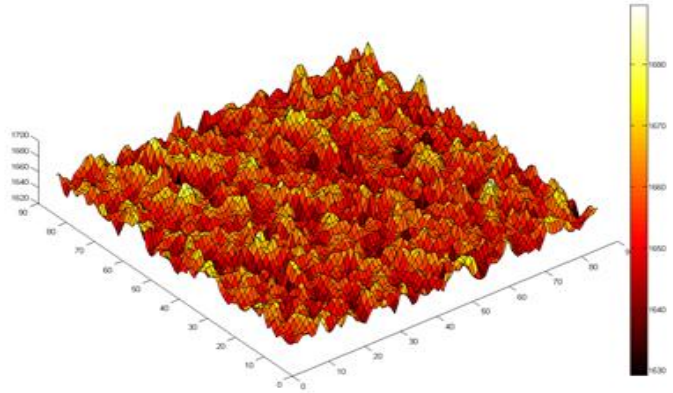
Figure 5.1 Average PSNR values for the four sequences using ES and PSO [52] using different resolutions.



(a) 240p



(b) 480p



(c) 720p

Figure 5.2 3D plot of the MSE over the entire search area of a block from the Parkrun sequence.

5.3 Proposed Hybrid Motion Estimation Algorithm

In this section, the steps of the proposed hybrid PSO motion estimation algorithm are

presented. For each candidate block in the current frame, a swarm of PSO particles is first initialized using the initialization scheme given in (3.11). In each iteration, particles update their positions and velocities using (2.4) and (2.5). The maximum allowed velocity of the PSO particles is adaptively-varied in each iteration of the process according to (2.9). Fitness values for the new positions are evaluated following the guidelines of the FFHP scheme proposed in section 2.2.2.2 . Then, particles are ranked according to the fitness function of their best positions P_{best} and certain particles are removed and others selected as parents using a novel selection scheme. Offsprings are then generated from the selected parents using a novel crossover operator. Mutation is then performed on the PSO particles. The whole scheme ends when termination conditions are satisfied. The details of these steps are explained as follows.

5.3.1 Selection

The selection operator in GA is responsible for ensuring survival of the best fitted individuals in the population. Selection is integrated in our proposed hybrid scheme to prevent PSO from wasting resources on weak individuals. In the proposed scheme, in each iteration, particles are first ranked according to the fitness values of their personal best positions P_{best}. The personal best fitness values of the particles are used here rather than the current fitness values since a strong particle with a good history might happen to cross a weak position during its search and would end up being removed from the population. It frequently happens that any continuous path connecting two very close local optimizers might necessarily cross regions in which the objective function is very high. As a consequence, points which are very close to good ones might have large objective function values and are discarded from further consideration. In order to take into account the “bumpiness” of the objective function, a better strategy would be that of evaluating the

quality of a single point on the basis of the objective function's behavior in a neighborhood of that point [118]. Therefore, the particle is judged to be strong or weak based not on its instantaneous fitness value but on its overall search performance. The last two weak particles that have the worst personal best fitness values are removed. They are replaced by two off springs that are generated by the crossover of two strong parents selected from the swarm. Selection is an important part of genetic algorithms since it affects significantly their convergence. The basic strategy follows the rule: The better fitted an individual, the larger the probability of its survival and mating. The most straightforward implementation of this rule is the so-called roulette-wheel selection [107]. This method assumes that the probability of selection is proportional to the best personal fitness of an individual. This is a stochastic algorithm and involves the following technique: the individuals are mapped to contiguous segments of a line, such that each individual's segment is equal in size to the fitness value of its P_{best}. A random number is generated and the individual whose segment spans the random number is selected [107].

5.3.2 Crossover

Crossover is a fundamental mechanism in the GA algorithm. It is used to generate offsprings from the selected parents. In our proposed hybrid approach, a modified version of the Velocity Propelled Averaged Crossover (VPAC) proposed in [119]. The goal in [119] was to create two child particles whose position is between the parent's positions, but accelerated away from the parent's current direction (negative velocity) in order to increase diversity in the population. Towards the end of a typical PSO run, the population tends to be highly concentrated in a small portion of the search space, effectively reducing the search space. With the addition of the VPAC crossover operator, a portion of the population is always pushed away from the group,

increasing the diversity of the population and preventing immature convergence. In MVPAC, instead of using the parents' positions, the personal best positions attained so far by the parents are used instead. The positions of the generated offsprings using MVPAC are given by:

$$x_{c_1}(t) = \frac{P_1(t)+P_2(t)}{2.0} - k_1 * v_1(t), \quad (5.4)$$

$$x_{c_2}(t) = \frac{P_1(t)+P_2(t)}{2.0} - k_2 * v_2(t). \quad (5.5)$$

where $x_{c_1}(t)$ and $x_{c_2}(t)$ are the positions of the two generated children $c_1$and $c_2$respectively. $P_1(t)$ and $P_2(t)$ are the personal best positions of parents 1 and 2, and $v_1(t)$ and $v_2(t)$ are the current velocities of parents 1 and 2 respectively. $k_1$and k_2 are two uniform random variables in the range [0,1]. The children velocities are assigned to zero to re-initialize the search and their personal best positions, $P_{c_1}(t)$ and $P_{c_2}(t)$, are assigned to their new positions restarting the children's memory, as follows:

$$P_{c_1}(t) = x_{c_1}(t), \quad (5.6)$$

$$P_{c_2}(t) = x_{c_2}(t). \quad (5.7)$$

5.3.3 Mutation

The purpose of mutation is to diversify the search direction and prevent convergence to the local optimum. Mutation is a genetic operator that alters one or more gene values in a chromosome from its initial state. This can result in entirely new gene values being added to the gene pool. With these new gene values, the genetic algorithm may be able to arrive at better solution than was previously possible. Mutation is utilized in the proposed hybrid algorithm to further increase the diversity of the PSO particles and prevent the population from stagnating at any local optima. Mutation is applied to the particles positions in each iteration according to a user-definable mutation probability P_m. A particle chosen for mutation undergoes a random shift

of its position within an n-dimensional hypercube of side length equal to $2 * d$ centered on the current particle's position according to the following equation:

$$x'_i(t) = x_i(t) - d + 2 * d * rand(0,1), \quad (5.8)$$

where $x_i(t)$ is the position of particle i at iteration t. The value of d is taken as 10% of the search range p. Such a mutation operator can be viewed as a transition from a current solution to its neighborhood solution in local search algorithms [120].

The proposed PSO process terminates whenever the maximum number of iterations *N* is reached. Early termination of search is allowed whenever the fitness value of the global best position is less than a predefined threshold value T_{th}.

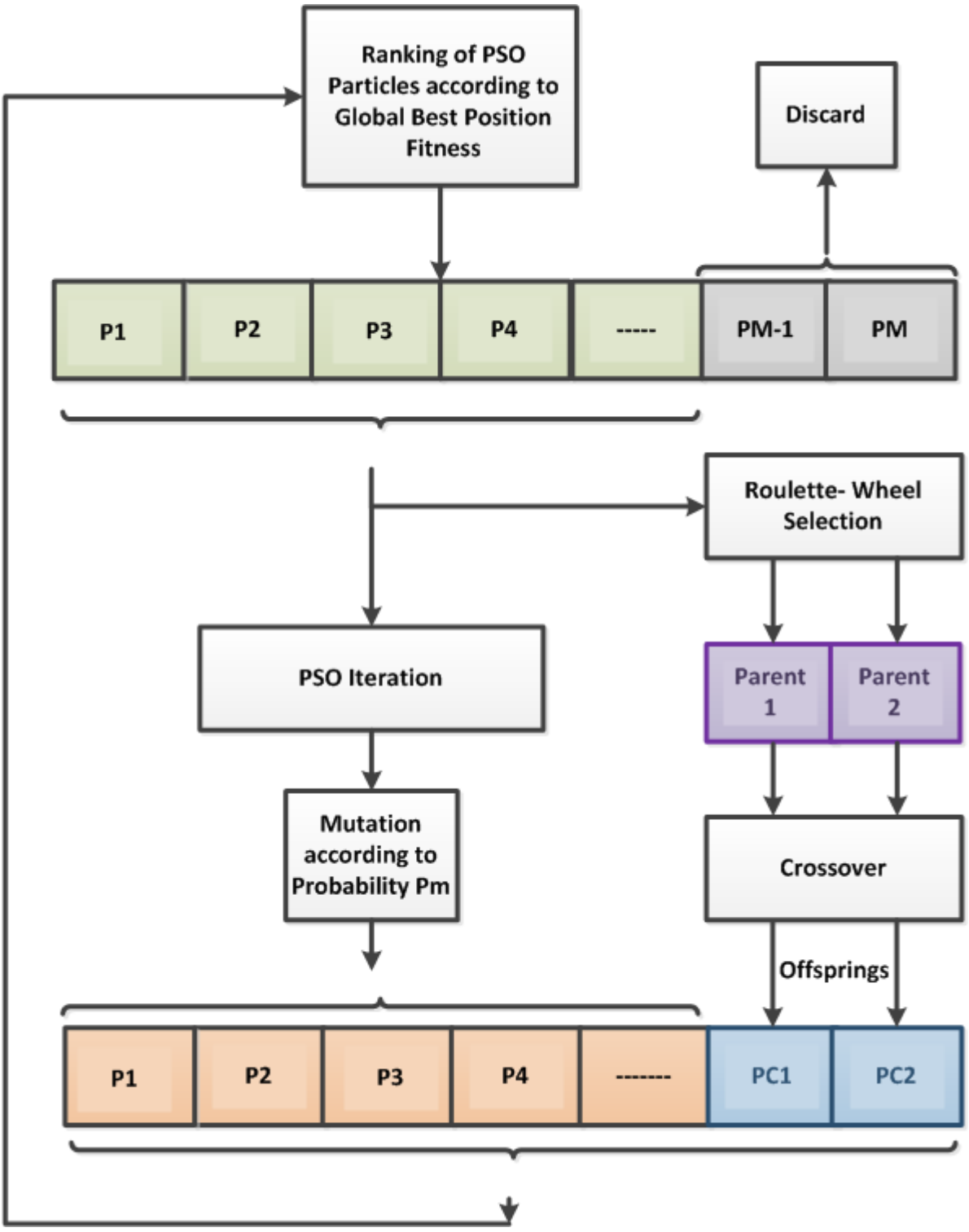


Figure 5.3 Flow of the Proposed Hybrid PSO-GA algorithm

5.4 Simulation Results and Performance Analysis

5.4.1 Search Precision

The performance of the proposed motion estimation algorithm was evaluated and compared in terms of accuracy and speed with the ES algorithm as well as several well-known fast search methods, including the four step search algorithm (4SS) [18] and the diamond search algorithm(DS) [19]. The results of the recent PSO-based motion estimation algorithm proposed in [52] are also included. The Peak Signal-to-Noise Ratio (PSNR) is used to measure the accuracy of motion estimation.

Simulations were conducted on a PC with Intel Core 2 Duo CPU at 2.26 GHz processor, 4GB RAM, and the MS Windows 7 OS. The source codes were written in Matlab 7.10. In the simulations, we used the first 100 frames of four HR 720p video sequences which are: Shields, Parkrun, Stockholm, and Mobcal. These sequences have been resized to two lower resolutions: 480p (854x480) and 240p (427x240). The block size used is 16 and the search area p is decreased proportionally with the resolution, where p is taken as 42, 24 and 12 for the 720p, 480p, and 240p resolutions respectively. The other parameters of simulation are as follows. For PSO, the size of the particle population was chosen to be M=10, N =4, T_{th}=7, c1 and c2 are equal to 2.05, and the fitness function is the MSE. The mutation probability P_m is taken as 20%.The average PSNR of each algorithm and the difference value between these search methods and ES are shown in Table 5.1. Fig.5.4 and Fig.5.5 also show the average PSNR values for the first 100 frames of the Parkrun and Mobcal sequences in the HD 720p resolution.

Table 5.1 Improvements in motion estimation quality interms of PSNR over the FS algorithm of the proposed hybrid PSO-GA algorithm as compared to existing techniques.

Algorithm		ES		4SS		DS		PSO[52]		Proposed	
Sequence	**Resolution**	PSNR	D-value	PSNR	D-value	PSNR	D-value	PSNR	D-value	PSNR	D-value
Parkrun											
	720p	25.628	0	23.667	1.9612	23.314	2.314	25.465	0.1632	25.565	0.0626
	480p	26.518	0	26.236	0.2817	26.298	0.2201	26.392	0.1262	26.467	0.0511
	240p	27.872	0	27.507	0.365	27.819	0.0522	27.791	0.0804	27.853	0.0185
Stockholm											
	720p	31.146	0	30.552	0.5937	30.528	0.6184	30.975	0.1707	31.026	0.1197
	480p	31.427	0	31.212	0.2151	31.217	0.2099	31.256	0.171	31.326	0.1008
	240p	32.259	0	31.500	0.7591	32.209	0.0508	31.916	0.3438	32.243	0.0167
Shields											
	720p	31.270	0	22.329	8.9413	23.575	7.695	31.010	0.2607	31.104	0.1667
	480p	31.012	0	25.201	5.8109	24.918	6.0942	30.775	0.2372	30.885	0.127
	240p	29.616	0	29.192	0.4237	29.175	0.4407	29.488	0.1282	29.560	0.0557
Mobcal											
	720p	30.323	0	23.217	7.1063	23.355	6.9682	30.068	0.2552	30.217	0.1059
	480p	30.411	0	25.430	4.9813	25.709	4.7021	30.183	0.2279	30.327	0.0841
	240p	31.299	0	31.170	0.1282	31.162	0.1369	31.168	0.1303	31.256	0.0428

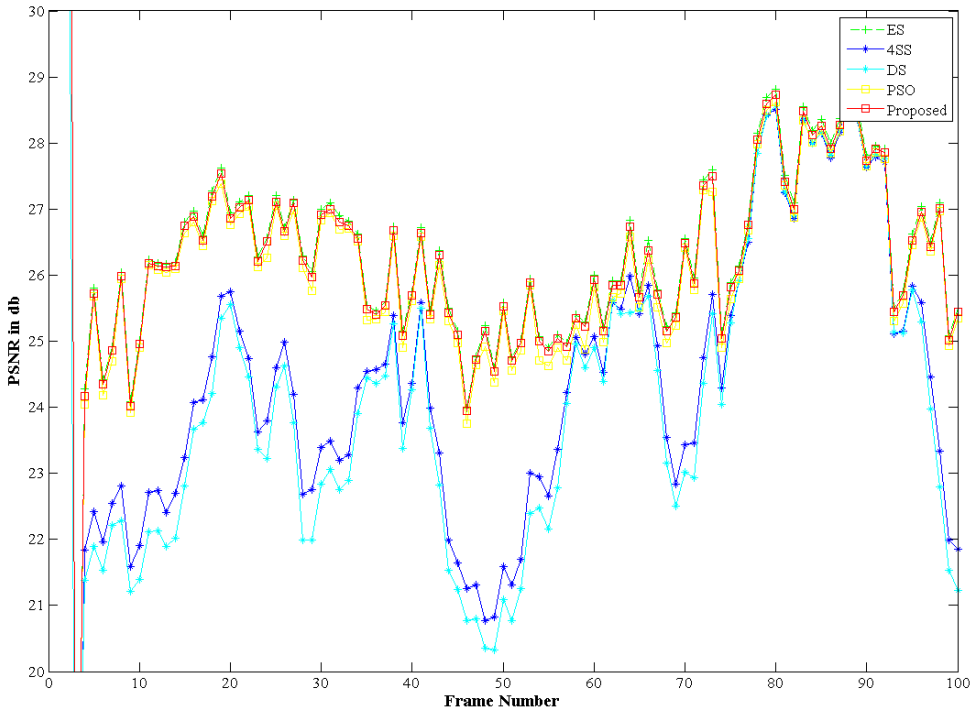


Figure 5.4 Average PSNR values for the first 100 frames of the Parkrun sequence in the 720p resolution.

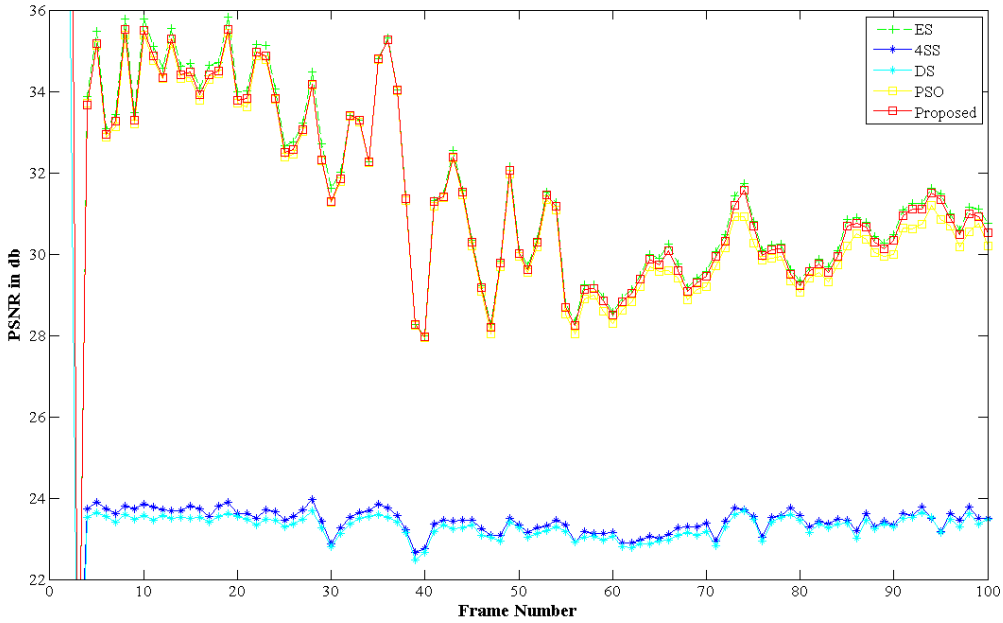


Figure 5.5 Average PSNR values for the first 100 frames of the Mobcal sequence in the 720p resolution.

5.4.2 Computational Complexity

In block matching motion estimation, the average number of candidate blocks checked for each MB is used as the evaluation criterion of computation complexity. In this chapter, the average number of fitness function evaluations for each MB is used as a metric of the computational complexity. Simulation results are listed in Table 5.2.

As shown in Table 5.2, the average number of search points needed is 12.8 for the 720p resolution, 8.7 for the 480p resolution, and 3.9 for the 240p resolution. Theoretically, for N=4, and M=10, the maximum number of fitness function evaluations is 40, but as shown in Table 5.2, the needed points are much less because of the effective strategies adopted in the proposed algorithm.

Table 5.2 Average number of fitness function evaluations per MB for the proposed hybrid algorithm based on the first 100 frames of each sequence.

Sequence	Resolution	Algorithm				
		ES	4SS	DS	PSO[52]	Proposed
Parkrun						
	720p	7225	22.2325	21.1811	16.0724	11.934
	480p	2401	19.6512	18.0614	15.1053	9.8434
	240p	625	18.0677	15.596	11.224	4.632
Stokholm						
	720p	6.77E+03	22.0487	20.9813	18.9818	14.9044
	480p	3.02E+03	19.5701	17.6164	12.2426	8.6044
	240p	758.728	18.5534	16.2674	5.8494	3.5372
Shields						
	720p	6.77E+03	29.8876	23.5864	15.8127	12.414
	480p	3.02E+03	25.2416	23.1129	12.726	9.2087
	240p	758.728	21.4554	20.7977	11.8026	5.1214
Mobcal						
	720p	6.77E+03	24.8717	23.0393	14.8885	11.933
	480p	3.02E+03	23.01	22.4502	10.1094	6.9518
	240p	758.728	18.8088	17.3101	5.1936	2.6239

5.5 Summary

In this Chapter, a hybrid PSO-GA algorithm is proposed for block motion estimation in HD video. High resolution videos tend to present very similar neighboring pixels and this fact contributes to increase the occurrence of local minima falls. The increase in the number of local minima enhances the problem of premature stagnation in the basic PSO algorithm. In this chapter, the strategies of the GA are incorporated into the PSO process to combat the problem of stagnation in HD video. A modified cross over operator is proposed for the breeding of the PSO particles. Simulation results demonstrate the superiority of the proposed scheme, in terms of computational complexity and motion estimation accuracy, as compared to existing algorithms.

CHAPTER 6

IMPLEMENTATION ON THE GPU

In this dissertation, parallel implementations of the proposed inter-MB and intra-MB algorithms on a shared memory multi-core CPU system using MATLAB PCT were presented in Chapters 2, 3, and 4. The NVIDIA programmable graphics processing unit (GPU) has evolved into a highly parallel, multithreaded, many-core processor with tremendous computational horsepower and very high memory bandwidth [121]. Modern GPUs can be found in virtually any relatively new computer. They are massively parallel processors designed to render millions of pixel values at a fraction of a second. Frameworks such as NVIDIA Compute Unified Device Architecture (CUDA) and Open Computing Language (OpenCL) allow supported GPUs to be used for general purpose programming. Combined with a general purpose processor such as an Intel Core i7, a modern GPU allows us to perform massively parallel computations on commodity hardware.

Therefore, the proposed algorithms are to be implemented on the NVIDIA GPU architecture using the CUDA platform. For the proposed cooperative parallel PSO algorithm in Chapter 2, it was shown that speedup of the parallel multi-core implementation is scalable with the video resolution. As a result, the parallel implementation of our proposed algorithm on the massively parallel architecture of modern GPUs, which consists of thousands of efficient cores, is expected to yield a tremendous improvement in performance.

In this Chapter, we present the parallel implementation of the cooperative PSO algorithm proposed in Chapter 2 on the NVIDIA GPU architecture using the CUDA platform. The rest of the chapter is organized as follows. In sections 6.1 and 6.2, we present an overview of the GPU and CUDA programming and memory models. In section 6.3, we present the details of the proposed

GPU implementation. Simulation results are given and analyzed in section 6.4. Finally, a summary is drawn in section 6.5.

6.1 The GPU

A GPU is a specialized hardware unit for rendering graphics on screen. It can typically be an integrated part of a motherboard chipset such as NVIDIA Ion [122], or as a discrete expansion card. In addition, modern processors such as AMD Fusion series [123] and Intel Sandy Bridge [124] Accelerated Processing Units (APUs) combine a CPU with a GPU on a single die, enabling more efficient communication between CPU and GPU [125].

While originally limited to rendering graphics, modern GPUs are essentially massively parallel processors. Designed to render 3D scenes onto a frame of 2D pixels, they enable the concurrent computation of large numbers of values. The first generations of GPUs had a fixed pipeline with limited programmability, but modern GPUs enable general purpose programming through C-like languages such as NVIDIA CUDA and OpenCL by the Khronos Group. Thus, the same thread model applied to pixel processing can be applied for solving problems not limited to graphics. This is known as general-purpose computing on graphics processing units (GPGPU), GPUs, due to their special purpose design, have a different architecture than CPUs.

CPUs spend much die space on control logic, such as branch prediction and out-of order execution and large cache to maximize performance [126]. GPUs have much less control logic, freeing up more die space for arithmetic logic units (ALUs). This gives a GPU more calculation capacity, at the cost of programming complexity. To reach peak performance, the programmer must explicitly design the application for the target GPU.

The computational strength of a GPU lies in performing the same calculations over a

large number of values. While originally limited to shaders performing transform and lightning of 3D graphics, the same processing power can be used for general purpose computations.

6.2 CUDA Programming model

Programming of the GPU follows the stream programming paradigm; the GPU code is implemented as kernels that get executed over the data. A kernel is written similarly as a regular sequential function, without any special vector instructions. It is then executed in one instance per thread by the CUDA schedulers. This is referred to as Single instruction, multiple threads (SIMT) model, as all the threads spawned from a single kernel call will issue the same instructions. The only differences between the threads are the special variables blockIdx and threadIdx. They identify the current thread, and get set at kernel invocation time. In addition, gridDim and blockDim will contain the maximum dimensions for the thread hierarchy.

6.2.1 Grid, blocks and threads

CUDA uses a two-tiered threading model that maps directly to the architecture of the GPU. Threads are bundled into groups, which are organized in a grid. The programmer is free to choose how the two tiers are organized, i.e. within the hardware limits and the compute capability of the GPU. The thread groups and grid may be organized as either a one-dimensional row, a two-dimensional grid, or a three dimensional cube. This is done with the dim3 integer vector types, which can contain up to three dimensions.

Prior to a kernel call, the programmer must specify the number and distribution of threads per block and blocks per grid. Figure 6.1 gives an example with a 2D grid containing 2D thread groups. The outer grid contains 6 thread groups, which each contains 12 threads.

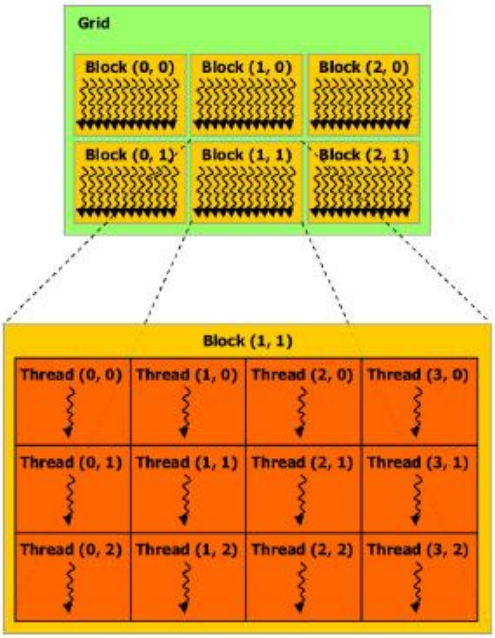


Figure 6.1 Example of an Execution Grid

As shown in Fig. 6.1, each thread within each block gets assigned a unique combination of dim3 blockIdx and threadIdx2. gridDim will be (3, 3, 1) and blockDim (3, 2, 1) These variables are then used to index the data set, effectively distributing the data among the threads.

6.2.2 CUDA Memory Model

The GPU has a memory hierarchy where memory usage can have a crucial impact on performance. Figure 6.2 shows the CUDA memory hierarchy. On the first level, we have the registers used by the CUDA cores. They have an access time of one clock cycle, but limited in size. Local memory is private to each thread, but resides in global memory which will be described below. The second level of memory is the shared memory, also residing on-chip. Shared memory also has access time of one clock cycle, but accessible to all the running threads of the same block. Off chip, the GPU has access to significantly larger amounts of memory, albeit with orders of

magnitude slower access times. This memory is referred to as global memory. CUDA architecture provides another kind of memory which we call Texture Memory. Like constant memory, texture memory is another variety of read-only memory that can improve performance and reduce memory traffic when reads have certain access patterns. Texture memory is located off-chip but is cached on chip, so in some situations it will provide higher effective bandwidth by reducing memory requests to off-chip DRAM. Although texture memory was originally designed for traditional graphics applications, it can also be used quite effectively in some GPU computing applications. When all threads in a warp are physically adjacent, using texture memory can reduce memory traffic and increase performance compared to global memory. The texture cache is optimized for 2D spatial locality, so threads of the same warp that read texture addresses that are close together will achieve best performance.

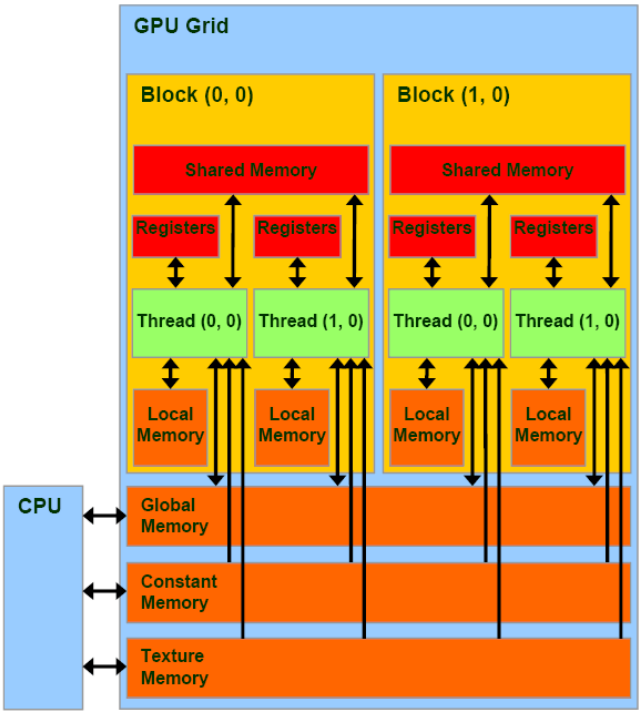


Figure 6.2 CUDA memory model

6.3 Proposed Parallel Implementation of the Cooperative PSO Algorithm Using CUDA

The parallel model of the proposed cooperative PSO algorithm for a QCIF video frame is shown in Figure 6.3. The steps of the parallel implementation are as follows:

6.3.1 Transferring Frames from the CPU to the GPU

The current frame and the previous frames are transferred from the CPU into the global memory of the GPU. The data transfer between CPU and GPU could be one of the major bottlenecks for achieving high performance. Taking the 720p video format as an example, for the PCI-E bus 2.0, the peak bandwidth is 8GB/s; the data transformation time of one frame is 0.12 ms. Therefore, an efficient implementation of this data transfer is needed. Higher bandwidth is possible between the host and the device when using page-locked (or "pinned") memory [127]. Host (CPU) data allocations are pageable by default. The GPU cannot access data directly from pageable host memory, so when a data transfer from pageable host memory to device memory is invoked, the CUDA driver must first allocate a temporary page-locked, or "pinned", host array, copy the host data to the pinned array, and then transfer the data from the pinned array to device memory. Pinned memory is used as a staging area for transfers from the device to the host. We can avoid the cost of the transfer between pageable and pinned host arrays by directly allocating our host arrays in pinned memory. Allocate pinned host memory in CUDA C/C++ using cudaMallocHost(), and deallocate it with cudaFreeHost().This is done as shown in Table 6.1. Space should then be allocated for the frames and the Pg array in the memory of the GPU. Data transfer is then performed by copying the frames from the pinned memory of the host to the GPU. Note that frames are stored in row-major order as one dimensional arrays in the memory of the GPU. The frames copied to the GPU are placed in the texture memory. This is performed in CUDA by first declaring the texture memory then binding the texture memory to the texture

reference. Reading from the texture memory is done via the texture reference in the kernel.

Table 6.1 CUDA code for transferring frames from CPU to the GPU

```
#define N 16; //Dimensions of the MB
#define WIDTH 176; //Width of a frame in a QCIF video sequence of the MB
#define HEIGHT 144; //Height of a frame in a QCIF video sequence of the MB
#define p 15; //Search area

int widthBlocks16 = WIDTH / N;
int heightBlocks16 = HEIGHT / N;
int numBlocks16x16 = widthBlocks16 * heightBlocks16;

//host memory allocation
PRINTF(("Allocating host memory."));
int frameSize = WIDTH * HEIGHT * sizeof(uint8_t);
uint8_t *h_frame1, *h_frame2;
checkCudaErrors (cudaMallocHost(&h_frame1, frameSize));
checkCudaErrors (cudaMallocHost(&h_frame2, frameSize));
memcpy(h_frame1, src1Data,frameSize);
memcpy(h_frame2, src2Data,frameSize);

//device memory allocation
PRINTF(("Allocating device memory."));
uint8_t *d_frame1, *d_frame2;
checkCudaErrors (cudaMalloc(&d_frame1, frameSize));
checkCudaErrors (cudaMalloc(&d_frame2, frameSize));
MV_t *Pg;
checkCudaErrors (cudaMalloc(&Pg, numBlocks16x16 * sizeof(MV_t)));

//copy current and reference frames to GPU
cudaMemcpy(d_frame1, h_frame1, frameSize, cudaMemcpyHostToDevice));
cudaMemcpy(d_frame2, h_frame2, frameSize, cudaMemcpyHostToDevice));

//create CUDA texture channel descriptor
cudaChannelFormatDesc textureDescU8;
textureDescU8 = cudaCreateChannelDesc<uint8_t> ();

//Bind device buffers to texture references (declared above kernel, must be in global scope) to enable caching
checkCudaErrors(cudaBindTexture2D(NULL, &frameRef, d_frame1, &textureDescU8, WIDTH, HEIGHT,
stride));
checkCudaErrors(cudaBindTexture2D(NULL, &frameCur, d_frame2, &textureDescU8, WIDTH, HEIGHT,
stride));
```

6.3.2 Setting up Execution Grid Parameters

The grid dimensions and block dimensions are defined depending on the resolution of the video sequence. For example, for a QCIF video format of 144*176 pixels, we have a total of 9*11 MBs each made up of 16*16 pixels. The number of blocks per grid is equal to the number of MBs in the video frame. The number of threads per block is equal to the dimension of the MB which is 16*16. Therefore, for a QCIF sequence, this is defined in CUDA as shown in Table 6.2. The proposed PSO ME kernel is then invoked using the defined parameters.

Table 6.2 CUDA code for defining grid parameters of the proposed GPU implementation

```
dim3 BlockThreads(N,N);
dim3 GridBlocks(WIDTH/N ,HEIGHT/N);
//Apply Motion Estimation Kernel
ME_PSO_GPU<<<GridBlocks,BlockThreads>>>(N, WIDTH, HEIGHT, p, Pg);
```

It is important to note here that designing the execution grid should be dependent on the targeted GPU architecture. It is important that the parameters of the execution grid are chosen to keep all CUDA cores busy, i.e. keeping a high occupancy of the GPU is important to harness the full computational power of the GPU. Increasing the number of threads to a high value may increase the extent of parallelism, but at the same time it will restrict the number of work groups that can be simultaneously active for concurrent execution in each multiprocessor. On the other hand, decreasing the number of threads to a small value increases the extent of serial execution within each thread. Thus, we need a tradeoff between the number of threads and the amount of computations by each thread [128]. NVIDIA Tesla C2050, for example, belongs to the Fermi architecture [129]. It has 14 streaming multiprocessors (SM) and each has a maximum of 1024

concurrently running threads, for a total of up to 14336 concurrently running threads across the entire GPU. The maximum thread blocks that can be launched simultaneously per SM is 8 which results in a total of 112. Choosing grid parameters of 16x16 threads per block and 9x11 blocks per grid for the QCIF sequence results in a total of 25344 threads which is more than the maximum number of threads that can run concurrently in the Tesla C2050, the number of blocks chosen to 99, however, is within the maximum block limit. In this case, CUDA runtime library will schedule the execution of the threads. Having a large number of threads per block has the advantage that all these threads have access to the shared memory of the block which has very fast access times. Another option for configuring the execution grid would be to choose 8x16=128 threads per block, where each thread will be responsible for two pixels in the MB, and 99 blocks per grid. This will result in a total of 12672, which is less than the maximum number of threads that can run concurrently on the GPU, and thus we wouldn't be using all the available processing capabilities of the GPU. However, we will ensure that all the issued threads are being executed simultaneously. For higher resolutions, CIF for example, 8x16 threads per block and 99 blocks per grid can also be used where in this case each thread block will be handling 4 MBs in the frame (every 32 threads responsible for one MB).

6.3.3 Proposed Kernel for the Cooperative PSO ME Algorithm

The details of the proposed kernel are as follows. In order to reduce the accessing to global memory, the pixels of an MB and its search window are first loaded to the shared memory of the thread-block so that they can be reused by all threads of the same thread-block. To do that, threads simultaneously fetch the pixels of the MB and the search area into the shared memory of the block. Memory coalescence, where consecutive threads access consecutive locations in the

memory, is used here to speedup global memory access.

Thread(0,0) in each block performs the first stage of PSO iterations sequentially. The threads within the block are utilized for parallel SAD calculation for each candidate point in the PSO search process. Parallel SAD calculation by the threads is performed using the reduction algorithm to speedup the calculations. At the end of the first stage of PSO iterations, thread(0,0) updates the value of Pg in the shared memory of the block as shown in Fig. 6.3.

Blocks then communicate the value of Pg to their neighbors via the global memory. Thread(0,0) of each block copies the attained value of Pg of its block to its corresponding location in the array in the global memory.

Thread(0,0) of each block then updates the particles of its swarm according to the received information and performs another round of PSO iterations.

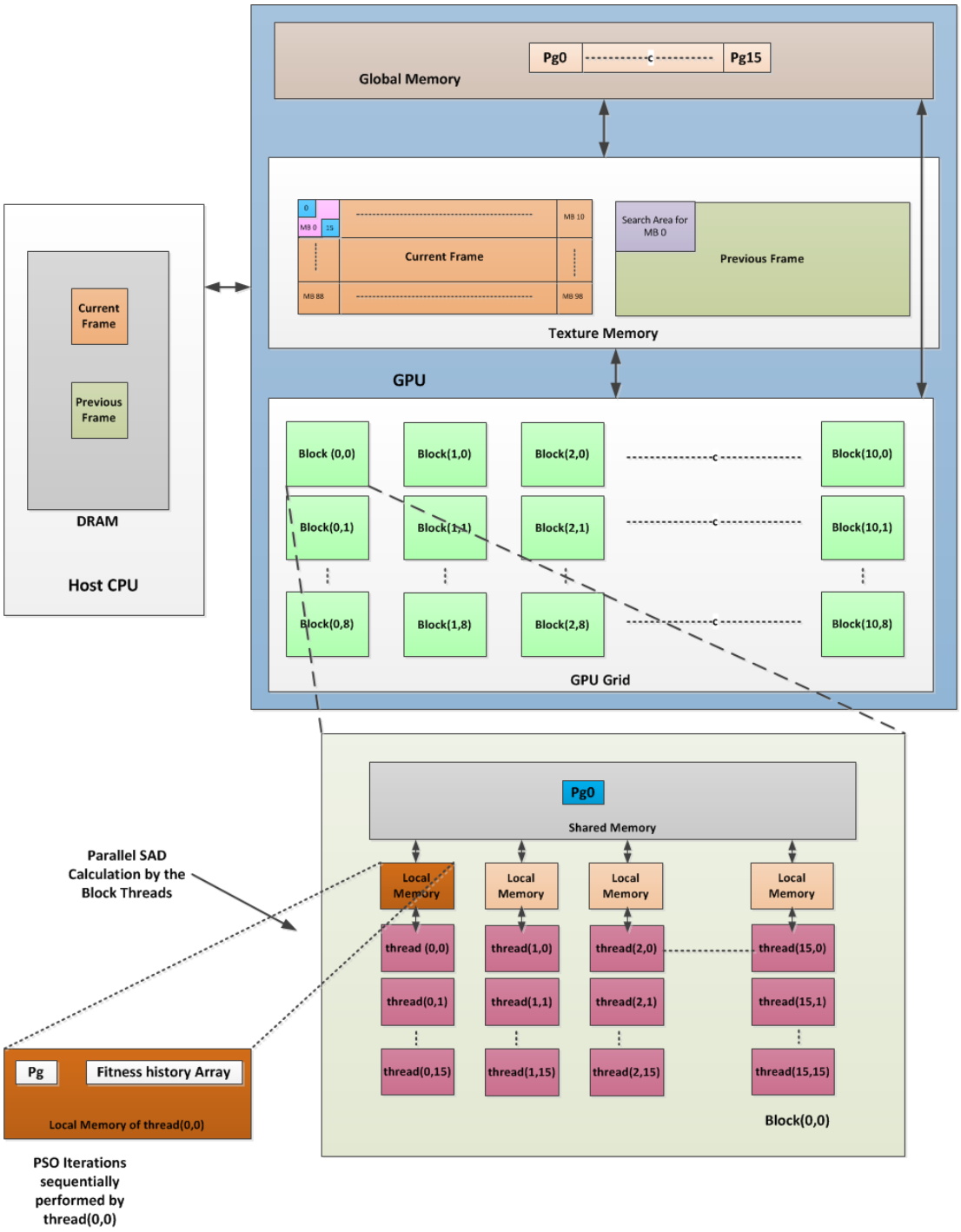


Figure 6.3 Parallel model of the proposed cooperative PSO algorithm on the GPU for a QCIF sequence

6.4 Simulation Results

The proposed parallel ME algorithm was tested on a host equipped with Intel CPU Intel

Xeon E5607 @ 2.27GHz. Tesla C2050 NVIDIA GPU is chosen the coprocessor to accelerate the proposed parallel scheme. The detailed information of the Tesla C2050 GPU can be seen in Table 6.3 [130]. The CUDA driver version used in our experiment was CUDA-7.2. Profiling the GPU implementations is performed using NVIDIA Visual Profiler.

The input videos in our experiment consist of a list of five standard test sequences in five resolutions: QCIF (Soccer), CIF (Bus), 480p (Racehorses), 720 p (Parkrun), and 1080 p (Pedestrian Area).

Table 6.3 Features of Tesla C2050

Compute Capability	2.0
Number of cores	448
Number of SM	14
Memory Bandwidth	144 GB/s
Frequency	1.15 GHz
Peak performance	1.03 Tflops
Maximum number of threads per block	1024
Maximum x-, y, or z dimension of a grid of thread blocks	65535
Maximum amount of shared memory per thread block	48 KB
Local memory per thread	512 KB
Constant memory size	64 KB

As was shown in Chapter 2, the parallel degree provided by the proposed cooperative PSO ME algorithm is equal to the number of MBs in a frame and is scalable with the video resolution as shown in table 6.4. We expect that the speedup will be proportional to the parallel degree of the video sequences. In table 6.5, we assess the performance of the proposed CUDA implementation on Tesla C2050. The average kernel execution times per frame and the average communication times between the CPU and GPU are given for the different video sequences. CPU-GPU (Host-to-Device) communication time is the time needed to transfer the video frame from the CPU (host) to the GPU (device), whereas the GPU-CPU (Device-to-Host) is the time

needed to copy the estimated motion vectors from the GPU back to the CPU. The execution time of the serial CPU implementation of the ES algorithm on Intel CPU Xeon E5607 is also given along with the execution times of the GPU kernel of the ES algorithm. Table 6.6 shows the achieved frame rate for the different implementations. As shown in table 6.6, motion estimation can be achieved at a rate that exceeds real time for the different resolutions. Fig. 6.4 shows the speedup achieved by the GPU implementations of the proposed cooperative PSO algorithm and the ES algorithm w.r.t the ES CPU implementation. As shown in Fig. 6.4, the achieved speedup is indeed scalable with the video resolution.

Table 6.4 Parallel degree of the proposed cooperative-PSO algorithm for the different video formats

Sequence	**Parallel Degree**
Soccer QCIF	99
Bus, CIF	396
RaceHorses, 480p	1560
Parkrun, 720p	3600
Pedestrian Area, 1080p	8040

Table 6.5 Performance of the proposed parallel implementation on Tesla C2050

Sequence	CPU-GPU Time (ms)	GPU-CPU Time (ms)	Proposed GPU Kernel (ms)	Total Time Proposed (ms)	ES GPU Kernel (ms)	ES GPU Total Time (ms)	ES CPU Time (ms)
Soccer QCIF, 15 fps, p=15	0.011	0.0018	0.863	0.875	4.3	4.31	36.09
Bus, CIF, 30 fps, p=15	0.031	0.0039	2.73	2.765	6.72	6.75	160.5
RaceHorses, 480p, 30 fps, p=15	0.082	0.0046	8.02	8.106	26.31	26.39	660.9
Parkrun 720p, 50 fps, p=15	0.176	0.0053	12.03	12.21	60.03	60.21	1559.5
Pedestrian Area 1080p, 50 fps, p=15	0.401	0.0080	16.901	17.310	131.78	132.19	3961.6

Table 6.6 Achieved frame rate in fps for the GPU implementations of the proposed approach and ES

Sequence	Proposed GPU implementation	ES GPU Implementation
Soccer QCIF, 15 fps, p=15	1142.47	231.89
Bus, CIF, 30 fps, p=15	361.68	148.04
RaceHorses, 480p, 30 fps, p=15	123.36	37.87
Parkrun, 720p, 50 fps, p=15	81.9	16.6
Pedestrian Area , 1080p, 50 fps, p=15	57.77	7.56

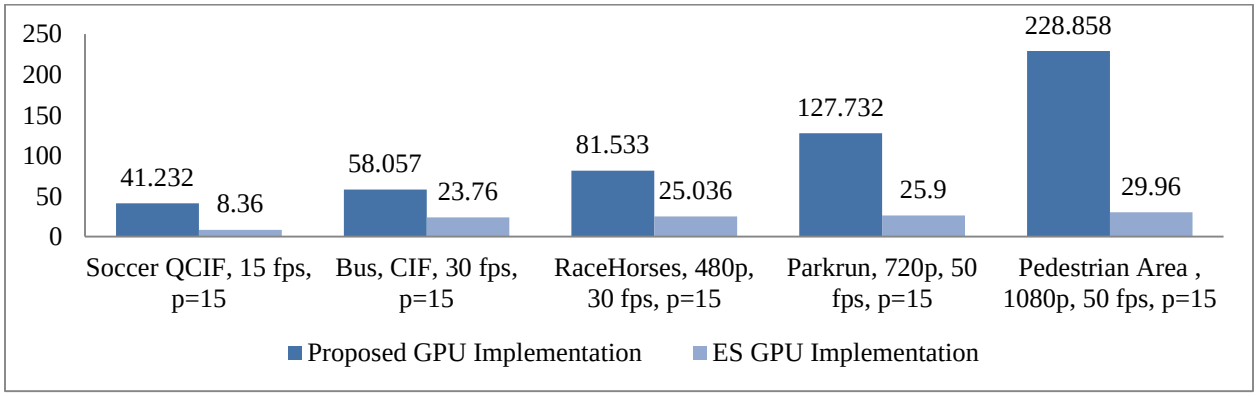


Figure 6.4 Comparison of the speedup achieved by the GPU implementations of the proposed cooperative PSO algorithm and ES

6.5 Summary

The parallel implementation of the proposed cooperative-PSO ME algorithm on the GPU architecture using CUDA is presented. The proposed implementation on the Tesla C2050 provides speedup scalable with the video resolution and satisfies the requirements of real time encoding of 50 fps for the 1080p resolution. Real time motion estimation for higher resolutions can be achieved with more advanced GPUs. The optimal parameters of the execution grid for the different video resolutions that would provide the highest speedup is currently being researched

CHAPTER 7

CONCLUSION

In this chapter, we summarize the main contributions of this thesis and indicate future directions to investigate which include direct extensions to the proposed work.

7.1 Contributions

In this thesis work, we have provided new contributions tackling the problem of block motion estimation in two areas: algorithm design and parallel implementation. These contributions are outlined as follows:

1- A cooperative PSO algorithm that achieves parallelism at the MB level is proposed in Chapter 2. Novel strategies are proposed to improve the accuracy and convergence speed of the PSO algorithm. It is found that the presented scheme provides improvements in terms of accuracy and computational complexity as compared to conventional fast motion estimation techniques and two state-of-the-art PSO-based ME schemes. An analysis of the parallel performance shows that the presented scheme is highly scalable and that the parallel efficiency increases with the increase in video resolution. The multicore implementation of the proposed algorithm using MATLAB PCT could achieve a speedup of 6.21 on eight CPU cores for HD video sequences. The multicore performance of the proposed scheme is also compared with existing parallel algorithms in the literature and is shown to give superior results.

2- A novel distributed game-theoretic approach to block motion estimation targeting

parallelism within the MB is proposed in Chapter 3. The optimization problem of BME of a given MB is cast in a game-theoretic setting using a network of autonomous players. It is shown that by using local communication and applying simple robust state-changing rules such as following natural game-theoretic dynamics, players can, in a distributed fashion, optimize the global objective function of the whole MB. Sequential and simultaneous algorithms based on BRD are proposed to solve the game in a distributed fashion. The efficiencies of the algorithms are demonstrated through both theoretical and simulation results. The analysis study show that our game-theoretic model is valid and presents a novel approach to BME compared to other classical methods, which is a kind of technology fusion of signal processing and AI. The multi-core implementation of the simultaneous scheme using MATLAB PCT shows that speedup is indeed obtained.

3- Parallelism within the MB is tackled again in Chapter 4 but from the view point of diffusion adaptation strategies in a multi-agent system. We formulate and study the distributed BME problem based on diffusion protocols to implement cooperation among individual adaptive agents. The individual agents are equipped with local learning abilities based on PSO. They derive local estimates for the motion vector and share information with their neighbors only, giving rise to peer-to-peer protocols. The resulting algorithm is distributed, cooperative, and inherently parallel. A diffusion-based PSO algorithm is proposed. The strategies of diffusion adaptation are incorporated into the PSO process by modifying the PSO velocity update equation and proposing a dynamically modified fitness function with regularization. The resulting algorithm is inherently parallel at the agents level within the MB. Simulation results show that the presented scheme satisfies the requirements of high estimation accuracy and low computational complexity while

achieving the targeted parallelism within the MB. The multi-core implementation of the algorithm using MATLAB PCT shows that speedup is obtained.

4- A novel hybrid PSO-genetic algorithm is proposed in Chapter 5 targeting BME in HR video. The strategies of crossover, mutation are adopted from the genetic algorithm and incorporated into the PSO process to combat the stagnation of the PSO particles in HR video. Simulation results show that the estimation accuracy is indeed improved relative to the basic PSO algorithm. The efficient strategies of particles initialization and fitness function history preservation maintain a low computational complexity for the proposed algorithm.

5- A parallel implementation of the cooperative PSO algorithm proposed in Chapter 2 on the NVIDIA GPU architecture using the CUDA platform is presented in Chapter 6. The MATLAB is an experimental computing resource and the MATLAB implementation of the proposed algorithm presented in Chapter 2 is intended to be used as a prototype to analyze the performance of the algorithm interms of estimation quality and computational complexity. The parallel implementation using MATLAB PCT also allows to analyze its parallel efficiency and scalability. However, the MATLAB implementation is not intended to be used a real time solution. In Chapter 6, a real time solution is presented by implementing the proposed algorithm on the NVIDIA Tesla C2050 GPU architecture using the CUDA platform.

7.2 Future Work and Possible Extensions

In this section, we present future extensions and interesting research directions that are worth investigating for the problem of BME.

7.2.1 A Unified Framework for Block Motion Estimation with Inter and Intra Block Parallelism

In Chapter 2 of this dissertation, we have presented an efficient cooperative multi-swarm PSO approach for BME that achieves parallelism between the MBs (inter-MB) of a given frame. In Chapters 3 and 4, game-theoretic and diffusion multi-agent BME frameworks that achieve fine-grained parallelism within the MB (intra-MB) were proposed. A natural extension to this work is to investigate a unified framework for BME with integrated multi –level parallelism. The investigated framework should achieve parallelism between, as well as, within the MBs of a given frame. This can be achieved by integrating the proposed intra-MB parallel algorithms into the proposed inter-MB parallel framework. This would result in a massively-parallel BME algorithm. The resulting scheme is to be implemented on the GPU and evaluated.

7.2.2 Macroblock Overlapping

The presented algorithms have used the basic model for block motion estimation which divides the frame into non-overlapping equally-sized blocks. A possible research direction is to investigate the effect of block overlapping on the quality and complexity of the proposed algorithms. On one hand, MB overlapping in the frame would increase the level of spatial correlation between neighboring MBs. As was mentioned in Chapter 2, the spatial correlation between neighboring MBs is exploited during the cooperation phase of the algorithm to improve the accuracy and convergence speed of the PSO algorithm. Therefore, increasing spatial correlation through MB overlapping is expected to lead to an improvement in the performance of the proposed cooperative PSO algorithm. Moreover, subblock overlapping within the MB can also have a direct effect on the proposed game-theoretic and diffusion algorithms. In the proposed

algorithms in Chapters 3 and 4, consensus between the players in a game (or agents in a multi-agent system) within the MB is targeted. Increasing the spatial correlation between neighboring subblocks is expected to speed up the consensus process. On the other hand, block overlapping would entitle increasing the size of the blocks which leads to an increase in the computational complexity of calculating the BDM. Intelligent algorithms, however, can be designed to overcome the potential increase in computational complexity.

7.2.3 Realistic Motion Model

The proposed BME algorithms proposed in this dissertation use the translational motion model. Specifically, the basic assumption in this technique is that the motion of all the pixels of each block is the same, more precisely, purely translational; and hence it can be described by only one vector per block. Clearly, this assumption is not realistic and as a result, simple translational model may fail to identify the actual movement in a video especially when there is complex object movement in the scene. There is a need to replace the conventional translational motion model with more robust and higher order models.

In order to achieve more accurate motion estimation without overly increasing computational demands, a number of techniques have been proposed, which generalize the block-based algorithms [131-134]. The movement of each block is rendered more realistically than in simple block-based algorithms by employing more complex spatial transformations such as the affine, perspective or bilinear transformation, or by employing elastic motion models which include the simple translation as a special case.

The proposed game-theoretic and diffusion PSO algorithms in this dissertation perform parallel motion estimation for the MB by dividing it into subblocks and then, simultaneously,

estimating the motion vectors of the subblocks. Such an approach can be used to develop a BME algorithm with a more realistic motion model. In fact, the estimated motion vectors of the subblocks can be successively used to estimate the parameters of the motion model.

7.2.4 Adaptively Weighted SAD Measure

The SAD measure is the commonly used BDM for the block motion estimation problem. In SAD, all pixels in the MB have equal weights. An interesting research direction would be to investigate the effect of using different weights for the pixels of the MB in the SAD measure during the search process. For example, some pixels in the MB are part of the background, while others lie on the edge of an object. Intuitively, a higher weight should be given to edge pixels during the ME process in order to improve the quality of the predicted video frame. It has been shown that weighting more the trajectories corresponding to sharp features than the trajectories with smooth texture leads to better reconstruction [135, 136]. Rather than using fixed weights, adaptively-changing weights can be used during the ME search process. Several effective filters have been designed in the literature for image edge detection that can be applied [137-138]. Considering a template MB at position (*x*, *y*) in the current frame and the candidate MB at position $(x + \hat{u}, y + \hat{v})$ in the previous frame I^{t-1}, the adaptive SAD (ASAD) measure would be:

$$ASAD(\hat{u}, \hat{v}) = \sum_{j=0}^{N-1} \sum_{i=0}^{N-1} \alpha_{ij}(k) * |g_t(x + i, y + j) - g_{t-1}(x + \hat{u} + i, y + \hat{v} + j)|, \quad (7.1)$$

where $g_t(.)$ is the gray value of a pixel in the current frame I^t and $g_{t-1}(.)$ is the gray level of a pixel in the previous frame I^{t-1}. $\alpha_{ij}(k)$ is an adaptive weight for pixel (i, j) in the MB that changes its value during each iteration k of the search process. This could greatly enhance the quality of the predicted frames.

7.2.5 Incorporating Color Information

So far, existing approaches for BME use the intensity (gray) information in the video frames and ignore the available color information during the motion search process. It is worth investigating if the ME algorithms presented can be extended by picking up suitable color(s) to track in the original frame and/or intelligently injecting color to the original frame. Some ideas on how to use color characteristics for motion detection were proposed in [139].

7.2.6 Deep Learning

Deep learning is a branch of artificial intelligence that lets computers solve problems that are too complex for conventional programming [140]. Training any deep learning system involves feeding it massive amounts of data. The clue to how deep learning works is in the name: systems learn from experience, much like people do. Thanks to its affinity with the parallel architecture of the graphics processing unit, deep learning is massively accelerated by GPUs [141]. Applying deep learning to the problem of block motion estimation is an interesting research direction worth investigating.

BIBLIOGRAPHY

[1] YouTube, “One Hour Per Second,” http://www.onehourpersecond.com/.

[2] ISO/IEC JTCl/SC29/WG11, “ISO/IEC CD 13818: Information technology,” MPEG-2 Committee Draft, Dec. 1993.

[3] International Telecommunication Union, “Video codec for audiovisual services at p x 64 kbits,” ITU-T Recommendation H.261, Mar. 1993.

[4] International Telecommunication Union, “Video coding for low bitrate communication,” ITU-T Draft H.263, July 1995.

[5] International Telecommunication Union, “Advanced Video Coding for Generic Audiovisual Services,” ITU-T Recommendation H.264, Nov. 2007.

[6] G.J. Sullivan, J.R. Ohm, W.J. Han, T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard," Circuits and Systems for Video Technology, IEEE Transactions on , vol.22, no.12, pp.1649-1668, Dec. 2012.

[7] Rao, K. R., Do Nyeon Kim, and Jae Jeong Hwang. "Video coding standards." *The Netherlands: Springer* (2014).

[8] Y. S. Cheng, Z. Y. Chen, and P. C. Chang, “An H.264 spatio-temporal hierarchical fast motion estimation algorithm for high-definition video,” in Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS '09), pp. 880–883, May 2009.

[9] Dimitrios Tzovaras, Ioannis Kompatsiaris, Michael G. Strintzis. 3D object articulation and motionestimation in model-based stereoscopic videoconference image sequence analysis and coding. SignalProcessing: Image Communication, 14(10), 1999, 817-840.

[10] Barron, J.L., Fleet, D.J., Beauchemin, S.S., 1994. Performance of optical flow techniques. Int. J. Comput.Vision 12 (1), 43–77.

[11] J. Skowronski. Pel recursive motion estimation and compensation in subbands. Signal Processing: ImageCommunication 14, (1999), 389-396.

[12] S. Metkar and S. Talbar, Motion Estimation Techniques for Digital Video Coding, SpringerBriefs in Computational Intelligence, chapter 2, 2013.

[13] Huang, T., Chen, C., Tsai, C., Shen, C., Chen, L. Survey on Block Matching Motion EstimationAlgorithms and Architectures with New Results. Journal of VLSI Signal Processing 42, 297–320, 2006.

[14] International Organization for Standardization. ISO/IEC 15938-5:2003: Information Technology—Multimedia Content Description Interface—Part 5: Multimedia Description Schemes, 1st edn. Geneva, Switzerland, 2003.

[15] H.-M. Jong, L.-G. Chen, and T.-D. Chiueh, "Accuracy improvement and cost reduction of 3-step searchblock matching algorithm for video coding," IEEE Trans. Circuits Syst. Video Technol., vol. 4, pp. 88–90,Feb. 1994.

[16] R. Li, B. Zeng, M.L. Liou, "A new three step search algorithm for block motion estimation," IEEE Trans. Circuits Syst. Video Technol.,vol.4, no.4, pp. 438–442, 1994.

[17] Jianhua Lu, and Ming L. Liou, "A Simple and Efficent Search Algorithm for Block-Matching MotionEstimation", IEEE Trans. Circuits And Systems For Video Technology, vol 7, no. 2, pp. 429-433, April 1997.

[18] L.M. Po, W.C. Ma, "A novel four-step search algorithm for fast blockmotion estimation," IEEE Trans. Circuits Syst. Video Technol., vol.6, no.3, pp. 313–317, 1996.

[19] S. Zhu, K. K. Ma, "A new diamond search algorithm for fast block-matching motion estimation," IEEE Transactions on Image Processing, vol. 9, pp. 287–290, 2000.

[20] C. H. Cheung, L. M. Po, "A novel cross-diamond search algorithm for fast block motion estimation," IEEE Transactions on Circuits and Systems for VideoTechnology 12 (12) (2002) 1168–1177.

[21] C. Zhu, X. Lin, and L. P. Chau, "Hexagon-based search pattern forfast block motion estimation," IEEE Trans. Circuits Syst. VideoTechnol., vol. 12, no. 5, pp. 349–355, May 2002.

[22] Yao Nie, and Kai-Kuang Ma, Adaptive Rood Pattern Search for Fast Block-Matching Motion Estimation, IEEE Trans. Image Processing, vol 11, no. 12, pp. 1442-1448, December 2002.

[23] Yi-Ching L., Jim L., Zuu-Chang H. Fast block matching using prediction and rejection criteria. Signal Processing, vol. 89, 2009, pp. 1115–1120.

[24] Liu, L., Feig, E. A block-based gradient descent search algorithm for block motion estimation in video coding, IEEE Trans. Circuits Syst. Video Technol., vol. 6, no. 4,1996,pp. 419–422.

[25] Saha, A., Mukherjee, J., Sural, S. A neighborhood elimination approach for block matching in motion estimation, Signal Process Image Commun, (2011), 26, 8–9, 2011, pp. 438–454.

[26] K.H.K. Chow, M.L. Liou, Generic motion search algorithm for video compression, IEEE Trans. Circuits Syst. Video Technol. , vol. 3, 1993, pp. 440–445.

[27] A. Saha , J. Mukherjee, S. Sural. New pixel-decimation patterns for block matching in motion estimation. Signal Processing: Image Communication 2008, vol. 23, pp. 725–738.

[28] Y. Song, T. Ikenaga, S. Goto. Lossy Strict Multilevel Successive Elimination Algorithm for Fast Motion Estimation. IEICE Trans. Fundamentals E90(4), 2007, 764-770.

[29] Z. B. Chen, P. Zhou, and Y. He, “Fast Integer Pel and Fractional Pel Motion Estimation for JVT,” in Proc. 6th Meeting: JVT–F017, Awaji Island, Japan, 2002.

[30] Humaira Nisar, Aamir Saeed Malik, Tae-Sun Choi. Content adaptive fast motion estimation based on spatio-temporal homogeneity analysis and motion classification. Pattern Recognition Letters, vol. 33, 2012, pp. 52–61.

[31] Zhiru Shi, Fernando, W.A.C., De Silva, D.V.S.X.., "A motion estimation algorithm based on Predictive Intensive Direction Searchfor H.264/AVC," IEEE Int. Conf. Multimedia and Expo (ICME), pp.667-672, July 2010.

[32] Zhiru Shi, W.A.C. Fernando and A. Kondoz, “An Efficient Fast Motion Estimation in H.264/AVC by Exploiting MotionCorrelation Character,”IEEE International Conference on Computer Science and Automation Engineering (CSAE), vol. 3, pp. 298 – 302, 25-27 May 2012.

[33] P. I. Hosur,“Motion Adaptive Search for Fast Motion Estimation,” IEEE Trans. Consumer Electronics, vol. 49, pp. 1330-1340, 2003.

[34] Ki Beom K, Young J, Min-Cheol H, “Variable Step Search Fast Motion Estimation for H.264/AVC Video Coder,” IEEE Trans. Consumer Electronics, vol. 54: pp. 1281-1286, 2008.

[35] Goel S and Bayoumi M. A, “Multi-Path Search Algorithm for Block-Based Motion Estimation,” IEEE Int. Conf Image Processing, pp. 2373-2376, 2006.

[36] P.I. Hosur and K.K. Ma, “Motion Vector Field Adaptive Fast Motion Estimation,” Second International Conference on Information, Communications and Signal Processing (ICICS ’99), Singapore, 7-10 Dec’99.

[37] A.M. Tourapis, O.C. Au, and M.L. Liou, "Predictive Motion Vector Field Adaptive Search Technique (PMVFAST) - Enhancing Block Based Motion Estimation,” in proceedings of Visual Communications and Image Processing 2001(VCIP-2001), pp.883-892, San Jose, CA, January 2001.

[38] A.M. Tourapis, O.C. Au, and M.L. Liou, "New Results on Zonal Based Motion Estimation Algorithms – Advanced Predictive Diamond Zonal Search," in proceedings of 2001 IEEE International Symposium on Circuits and Systems(ISCAS-2001), vol.5, pp.183–186, Sydney, Australia, May 6-9, 2001.

[39] A. M. Tourapis. "Enhanced predictive zonal search for single and multiple frame motion estimation." Electronic Imaging 2002.International Society for Optics and Photonics, pp. 1069-1079, 2002.

[40] L.T. Ho andJ.M. Kim, “Direction Integrated Genetic Algorithm for Motion Estimation in H.264/AVC,” Advanced Intelligent Computing Theories and Applications. With Aspects of Artificial IntelligenceLecture Notes in Computer Science,Vol. 6216, pp. 279-286, 2010.

[41] A. El Ouaazizi, M. Zaim, & R. Benslimane, “A Genetic Algorithm for Motion Estimation,” IJCSNS International Journal of Computer Science and Network Security, VOL.11 No.4, April 2011.

[42] Z. Shi, W.A.C. Fernando, and A. Kondoz,“Simulated Annealing for Fast Motion Estimation Algorithm in H.264/AVC,” Simulated Annealing - Single and Multiple Objective Problems, Marcos de Sales Guerra Tsuzuki (Ed.), ISBN: 978-953-51-0767-5, InTech, DOI: 10.5772/50974. Available from: http://www.intechopen.com/books/simulated-annealing-single-and-multiple-objective-problems/simulated-annealing-for-fast-motion-estimation-algorithm-in-h-264-avc.

[43] E. Cuevas, D. Zaldívar, M.P. Cisneros, H. Sossa, V. Osuna, “Block Matching Algorithm for Motion Estimation Based on Artificial Bee Colony (ABC),” Applied Soft Computing, vol. 13, issue 6, June 2013, pp. 3047-3059.

[44] E. Cuevas, D. Zaldívar, M. P. Cisneros, D. Oliva, “Block-Matching Algorithm Based on Differential Evolution for Motion Estimation,” Engineering Applications of Artificial Intelligence, vol. 26, issue 1, pp. 488-498, January 2013.

[45] G.Y. Du, T.S. Huang, L.X. Song, and B.J. Zhao, “A Novel Fast Motion Estimation Method Based on Particle Swarm Optimization,” Fourth International Conference on Machine Learning and Cybernetics, 2005.

[46] K.M. Bakwad, S.S. Pattnaik, B.S. Sohi, S. Devi, S. Gollapudi, C.V. Sagar, and P.K. Patra, “Small Population Based Modified Parallel Particle Swarm Optimization for Motion Estimation,” 16th International Conference on Advanced Computing and Communications (ADCOM’2008), 2008.

[47] R. Ren, M.M. Manokar, Y. Shi, B. Zheng, “A Fast Block Matching Algorithm for Video Motion Estimation Based on Particle Swarm Optimization and Motion Prejudgement,” 2006.

[48] X. Yuan and X. Shen, “Block Matching Algorithm Based on Particle Swarm Optimization for Motion Estimation,” in: International Conference on Embedded Software and Systems (ICESS’2008), 2008.

[49] Z. Ping, C. Hu, and W. Ping,“Fast Motion Estimation Algorithm for Scalable Motion Coding,” 2010 International Conference on Electrical and Control Engineering (ICECE), 25-27 June 2010.

[50] K.M. Bakwad, S.S. Pattnaik, B.S. Sohi, S. Devi, S. Gollapudi, V.R.S. Sastry, C.V. Sagar, and P.K. Patra, "Fast Motion Estimation using Small Population-Based Modified Parallel Particle Swarm Optimisation," IJPEDS, vol. 26, no. 6, pp. 457-476, 2011.

[51] S. Immanuel Alex Pandian, G. Josemin Bala, and J. Anitha, “A Pattern Based PSO Approach for Block Matching In Motion Estimation,” Engineering Applications of Artificial Intelligence, vol. 26, issue 8, pp. 1811‑1817, September 2013.

[52] J. Cai and W. David Pan, “On Fast And Accurate Block-Based Motion Estimation Algorithms Using Particle Swarm Optimization,” Information Sciences, vol. 197, pp. 53–64, 15 August 2012.

[53] J. Kennedy, R.C. Eberhart, Particle swarm optimization, in: Proceedings of the IEEE International Conference on Neural Networks, vol. IV, Perth,Australia, 1995.

[54] R.C. Eberhart, J. Kennedy, A new optimizer using particle swarm theory, in: Proceedings of the Sixth International Symposium on Micro Machine andHuman Science, 1995.

[55] J. Kennedy, R. Mendes, Neighborhood topologies in fully-informed and best-of-neighborhood particle swarms, in: Proceedings of the IEEE International Workshop on Soft Computing in Industry Applications, 2003.

[56] F. van den Bergh, A. Engelbrecht, A study of particle swarm optimization particle trajectories, Information Sciences 176 (8) (2006) 937–971.

[57] A. Heikkinen andL. Fono, "Parallel Implementations of Motion Estimation Algorithms Using OPENCL," Digital Signal Processing (DSP), 2013 18th International Conference on, 1-3 July 2013.

[58] J. Zhang,J.F. Nezan, and J.G. Cousin, "Implementation of Motion Estimation Based on Heterogeneous Parallel Computing System with OpenCL," High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on , pp.41-45, 25-27 June 2012.

[59] R. Cheng, E. Yang, and T. Liu, "Speeding Up Motion Estimation Algorithms on CUDA Technology," Microelectronics and Electronics (PrimeAsia), 2010 Asia Pacific Conference on Postgraduate Research in, pp. 93- 96, 2010.

[60] Z. Jing, J. Liangbao, and C. Xuehong, "Implementation of parallel full search algorithm for motion estimation on multi-core processors," Next Generation Information Technology (ICNIT), 2011 The 2nd International Conference on , vol., no., pp. 31-35, 21-23 June 2011.

[61] E. Monteiro, B. Vizzotto, C. Diniz, M. Maule, B. Zatt, andS. Bampi ,“Parallelization of Full Search Motion Estimation Algorithm for Parallel and Distributed Platforms,” International Journal of Parallel Programming, April 2014, Volume 42, Issue 2, pp 239-264.

[62] V. Dung, Y. Yang, and B. Laxmi, "An Efficient Dynamic Multiple-Candidate Motion Vector Approach for GPU-Based Hierarchical Motion Estimation," Performance Computing and Communications Conference (IPCCC), pp.342-351, 2012.

[63] The Mathworks Inc, Parallel Computing Toolbox User’s Guide. http://www.mathworks.com/products/parallel-computing.

[64] Y. Shi, R.C. Eberhart, "Empirical Study of Particle Swarm Optimization," Evolutionary Computation, 1999, CEC 99. Proceedings of the 1999 Congress on, vol.3, 1999.

[65] X. Li, N. Xiao, C. Claramunt, and H. Lin, “Initialization Strategies to Enhancing the Performance of Genetic Algorithms for the P-Median Problem”, Journal of Computers and Industrial Engineering, vol. 61, issue 4, pp. 1024-1034, Nov. 2011.

[66] N. Xiao, “A Unified Conceptual Framework for Geographical Optimization Using Evolutionary Algorithms,” Annals of the Association of American Geographers, vol. 98, issue 4, pp. 795–817, 2008.

[67] A. Grama, A. Gupta, G. Karypis, and V. Kumar, Introduction to Parallel computing, 2nd edition, Addison-Wesley, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2002.

[68] https://media.xiph.org/video/derf/.

[69] http://videocoders.com/yuv.html.

[70] J.L. Gustafson, “Reevaluating Amdahl's Law,”Communications of the ACM, vol. 31, no. 5, pp. 532-533, May 1988.

[71] Cortez and S. Martínez, “Self-triggered Best-Response Dynamics for Continuous Games”, IEEE Transactions on Automatic Control, vol. 60, issue 4, July 30 2014, pp. 1115 – 1120.

[72] D. Monderer and L. Shapley, “Potential games,” Games and economic behavior, vol. 14, pp. 124–143, 1996.

[73] Z. Lin and H.T. Liu, “Consensus Based on Learning Game Theory”, Guidance, Navigation and Control Conference (CGNCC), 2014 IEEE Chinese , 8-10 Aug. 2014.

[74] D. Fudenberg and J. Tirole, Game theory. MIT press Cambridge, Massachusetts, 1991.

[75] G. Scutari, D. Palomar, and S. Barbarossa, “Asynchronous Iterative water-filling for gaussian frequency-selective interference channels,” IEEE Transactions on Information Theory, vol. 54, no. 7, pp. 2868–2878, 2008.

[76] S. Lasaulce and H. Tembine, Game Theory and Learning for Wireless Networks: Fundamentals and Applications, Elsevier, Ed. New York, NY,USA: Academic Press, 2011.

[77] G. Bacci, W. Saad, S. Lasaulce and L. Sanguinetti, "Game Theory for Networks: A tutorial on game-theoretic tools for emerging signal processing applications," in *IEEE Signal Processing Magazine*, vol. 33, no. 1, pp. 94-119, Jan. 2016.

[78] Y. Wu and K. Liu, “An Information Secrecy Game in Cognitive Radio Networks,” IEEE Trans. Information Forensics and Security, vol. 6, no. 3, pp. 831–842, Sep. 2011.

[79] B. Wang, K. Liu, and T. Clancy, “Evolutionary Cooperative Spectrum Sensing Game: How to Collaborate?” IEEE Trans. Commun., vol. 58, no. 3, pp. 890–900, Mar. 2010.

[80] Jason R. Marden, GürdalArslan, and Jeff S. Shamma, “Cooperative Control and Potential Games”, IEEE transactions on Systems, Man, and Cybernetics—part b: Cybernetics, vol. 39, no. 6, December 2009.

[81] K. Yamamoto, “A Comprehensive Survey of Potential Game Approaches to Wireless Networks*,” IEICE Transactions on Communications 2015,vol.* 98, no.9,*pp.* 1804–1823.

[82] P. Coucheney, S. Durand, B. Gaujal, C. Touati, “General Revision Protocols in Best Response Algorithms for Potential Games,” IEEE Netwok Games, Control and Optimization (NetGCoop), Trento, Italy, Oct 2014.

[83] M. Wooldridge, An introduction to multiagent systems, 2nd edition. Hoboken, NJ, USA: Wiley, July 2009.

[84] J. Enright, K. Savla, and E. Frazzoli, “Coverage control for nonholonomic agents,” in Proc. IEEE Conf. on Decision and Control, Cancun,Mexico, Dec. 2008.

[85] A. Arsie, K. Savla, and E. Frazzoli, “Efficient routing algorithms for multiple vehicles with no explicit communications,” IEEE Trans. On Automatic Control, vol. 54, no. 10, pp. 2302–2317, Oct. 2009.

[86] O. Shehory and S. Kraus, “Task allocation via coalition formation among autonomous agents,” in Proc. of the Fourteenth International Joint Conference on Artificial Intelligence, Aug. 1995, pp. 655–661.

[87] O. Shehory and S. Kraus, “Methods for task allocation via agent coalition formation,” Artifical Intelligence Journal, vol. 101, no. 1, pp. 165–200, May 1998.

[88] B. Gerkey and M. J. Mataric, “A formal framework for the study of task allocation in multi-robot systems,” International Journal of Robotics Research, vol. 23, no. 9, pp. 939–954, Sept. 2004.

[89] M. Alighanbari and J. How, “Robust decentralized task assignment for cooperative UAVs,” in Proc. of AIAA Guidance, Navigation, and Control Conference, Colorado, USA, Aug. 2006.

[90] D. M. Stipanovic, P. F. Hokayem, M. W. Spong, and D. D. Siljak, “Co- operative avoidance control for multi-agent systems,” ASME Journal of Dynamic Systems, Measurement, and Control, vol. 129, no. 5, pp. 699–706, Sept. 2007.

[91] J. Yang and Z. Luo, “Coalition formation mechanism in multi-agent systems based on genetic algorithms,” Applied Soft Computing, vol. 7, no. 2, pp. 561–568, Mar. 2007.

[92] Q. Chen, M. Hsu, U. Dayal, and M. Griss, “Multi-agent cooperation, dynamic workflow and XML for e-commerce automation,” in Proc. Int. Conf. on Autonomous agents, Catalonia, Spain, June 2000.

[93] C. G. Lopes and A. H. Sayed. Diffusion least-mean squares over adaptive networks: Formulation and performance analysis. IEEE Transactions on Signal Processing, 56(7):3122–3136, Jul. 2008.

[94] F. S. Cattivelli and A. H. Sayed. Diffusion LMS strategies for distributed estimation. IEEE Transactions on Signal Processing, vol. 58, iss. 3, pp. 1035–1048, Mar. 2010.

[95] Chen, Jianshu, and Ali H. Sayed. "Diffusion adaptation strategies for distributed optimization and learning over networks." Signal Processing, IEEE Transactions on 60.8 (2012): 4289-4305.

[96] Jianshu Chen; Sheng-Yuan Tu; Sayed, A.H., "Distributed optimization via diffusion adaptation," Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP), 2011 4th IEEE International Workshop on , vol., no., pp.281,284, 13-16 Dec. 2011.

[97] Sayed, Ali H. "Diffusion adaptation over networks." arXiv preprint arXiv:1205.4220 (2012).

[98] F. Cattivelli and A. H. Sayed, "Modeling bird flight formations using diffusion adaptation," IEEE Transactions on Signal Processing, vol. 59, no. 5, pp. 2038–2051, May 2011.

[99] J. Li and A. H. Sayed, "Modeling bee swarming behavior through diffusion adaptation with asymmetric information sharing," EURASIP Journal on Advances in Signal Processing, 2012.

[100] Tu, Sheng-Yuan, and Ali H. Sayed. "Foraging behavior of fish schools via diffusion adaptation." Cognitive Information Processing (CIP), 2010 2nd International Workshop on. IEEE, 2010.

[101] C. G. Lopes and A. H. Sayed, "Distributed processing over adaptive networks," in Proc. Adaptive Sensor Array Processing Workshop, MIT Lincoln Laboratory, MA, June 2006, pp. 1–5.

[102] C. Lopes and A. Sayed, "Diffusion least-mean squares over adaptive networks," in IEEE ICASSP, vol. 3, Honolulu, HI, Apr. 2007, pp. 917–920.

[103] A. H. Sayed and C. G. Lopes, "Adaptive processing over distributed networks," IEICE Trans. Fund. Electron., Commun. Comput. Sci., vol. E90-A, no. 8, pp. 1504–1510, Aug. 2007.

[104] F. S. Cattivelli and A. H. Sayed, "Diffusion LMS algorithms with information exchange," in Proc. Asilomar Conf. Signals, Syst. Comput., Pacific Grove, CA, Nov. 2008, pp. 251–255.

[105] F. S. Cattivelli, C. G. Lopes, and A. H. Sayed, "A diffusion RLS scheme for distributed estimation over adaptive networks," in Proc. IEEE Workshop on Signal Process. Advances Wireless Comm. (SPAWC), Helsinki, Finland, June 2007, pp. 1–5.

[106] S. Li, W. Xu, N. Zheng, et al., "A Novel fast motion estimation method based on genetic algorithm," Acta Electronica Sinica, vol. 6, no. 28, pp. 114–117, 2000.

[107] D.E. Goldberg, Genetic Algorithms in Search, Optimization & Machine Learning Addison-Wesley, Reading, MA (1989).

[108] Z. Michalewicz, Genetic Algorithms+Data Structures=EvolutionPrograms. New York: Springer-Verlag, 1999.

[109] J. Arabas, Z. Michalewicz, and J. Mulawka, "GAVaPS—A genetic algorithm with varying population size," in Proc. IEEE Int. Conf. on Evolutionary Computation, Orlando, 1994, pp. 73–78.

[110] R. Tanese, "Distributed genetic algorithm," in Proc. Int. Conf. Genetic Algorithms, 1989, pp. 434–439.

[111] R. J. Collins and D. R. Jefferson, "Selection in massively parallel geneticalgorithms," in Proc. Int. Conf. Genetic Algorithms, 1991, pp. 249–256.

[112] D. E. Moriarty and R. Miikkulainen, "Efficient reinforcement learning through symbiotic evolution," Mach. Learn., vol. 22, pp. 11–32, 1996.

[113] D. Whitely, S. Dominic, R. Das, and C. W. Anderson, "Genetic reinforcement learning for neurocontrol problems," Mach. Learn., vol. 13, pp. 259–284, 1993.

[114] D. Thierens, "Adaptive mutation rate control schemes in genetic algorithms,"in Proc. IEEE Int. Conf. Evolutionary Computation, HI, 2002, pp. 980–985.

[115] S. Tsutsui and D. E. Goldberg, "Simplex crossover and linkage identification: Single-stage evolution vs. multi-stage evolution," in Proc. IEEE Int. Conf. Evolutionary Computation, HI, 2002, pp. 974–979.

[116] Gustavo Sanchez, Felipe Sampaio, Marcelo Porto, Sergio Bampi, and Luciano Agostini, "DMPDS: A Fast Motion Estimation Algorithm Targeting High Resolution Videos and Its FPGA Implementation," International Journal of Reconfigurable Computing, Article ID 186057, 12 pages, 2012.

[117] Andrei Lihu, ŞtefanHolban, "Particle Swarm Optimization with Disagreements on Stagnation," Semantic Methods for Knowledge Management and Communication, Studies in Computational Intelligence, vol. 381, 2011, pp. 103-113.

[118] Marco Locatelli, Fabio Schoen, "Global Optimization Based on Local Searches," 4OR, December 2013, Volume 11, Issue 4, pp. 301-321.

[119] Matthew Settles and Terence Soule, "Breeding swarms: a GA/PSO Hybrid," GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation, 2005.

[120] M.F. Bramlette, "Initialization Mutation and Selection Methods in Genetic Algorithms for Functions Optimization," Proceedings of the ICGA 4 (1991), pp.100–107

[121] "NVIDIA CUDA Compute Unified Device Architecture, Programming Guide version 2.0", 2008, found on www.nvidia.com.

[122] nVidia Corporation, "nvidia ion specifications." http://www.nvidia.com/object/picoatom_specifications.html.

[123] N. Brookwood, "Amd white paper: Amd fusion™ family of apus." http://sites.amd.com/us/Documents/48423B_fusion_whitepaper_WEB.pdf, March 2010.

[124] I. Corporation, "Products (formerly sandy bridge)." http://ark.intel.com/ products/codename/29900.

[125] M. Doerksen, S. Solomon, and P. Thulasiraman, "Designing APU oriented scientific computing applications in openCL," in 13th IEEE International Conference on High Performance Computing & Communication, HPCC 2011, Banff, Alberta, Canada, September 2-4, 2011, pp. 587–592, IEEE, 2011.

[126] B. Kirk andW.meiW.Hwu, Programming Massively Parallel Processors: A Hands-on Approach (Applications of GPU Computing Series). Morgan Kaufmann, 2010.

[127] Mark Horis, "How to Optimize Data Transfers in CUDA C/C++", Nvidia, December 2012. https://devblogs.nvidia.com/parallelforall/how-optimize-data-transfers-cuda-cc/.

[128] Utsab Bose, Anup Kumar Bhattacharya, Abhijit Das,"GPU-Based Implementation of 128-Bit Secure Eta Pairing over a Binary Field", 6th International Conference on Cryptology in Africa, Cairo, Egypt, June 22-24, 2013.

[129] nVidia. NVIDIA Fermi Compute Architecture Whiltepaper, v1.1 edition.

[130] nVidia, "www.nvidia.com/tesla".

[131] Konstantopoulos, Charalampos. "A Parallel Algorithm for Motion Estimation in Video Coding Using the Bilinear Transformation." SpringerPlus , issue. 4, June 2015, pp. : 288.

[132] Mokraoui A, Munoz-Jimenez V, Astruc J-P, "Motion estimation algorithms using the deformation of planar hierarchical mesh grid for videoconferencing applications at low bit-rate transmission," Journal of Signal Process Syst. 2012, vol. 67, iss. 2, pp.167–185.

[133] Huang H, Woods JW, Zhao Y, Bai H, "Control-point representation and differential coding affine-motion compensation," IEEE Trans Circuits Syst Video Technol. 2013, 23(10), pp. 1651–1660.

[134] Muhit AA, Pickering MR, Frater MR, Arnold JF, "Video coding using fast geometry-adaptive partitioning and an elastic motion model," Journal of Visual Communication and Image Representation 2012, vol. 23, iss. 1, pp.31–41.

[135] Jianbo Shi and C. Tomasi, "Good features to track," Computer Vision and Pattern Recognition, 1994. Proceedings CVPR '94., 1994 IEEE Computer Society Conference on, Seattle, WA, 1994, pp. 593-600.

[136] P. M. Q. Aguiar and J. M. F. Moura, "Image motion estimation-convergence and error analysis," Image Processing, 2001. Proceedings. 2001 International Conference on, Thessaloniki, 2001, pp. 937-940 vol.2.

[137] M. A. Al-Alaoui, " Direct Approach to Image Edge Detection Using Differentiators ," proceedings of the 17th IEEE International Conference on Electronics, Circuits, and Systems, Athens, Greece, pp.154-157, December 12-15, 2010.

[138] M. A. Al-Alaoui, "Novel FIR Approximations of IIR Differentiators with Applications to Image Edge Detection," proceedings of the 18th IEEE International Conference on Electronics, Circuits, and Systems, Beirut, Lebanon, pp. 554-558, December 11-14, 2011.

[139] M. A. Nehme, W. Khoury, B. Yameen and M. A. Al-Alaoui, "Real time color based motion detection and tracking," Signal Processing and Information Technology, 2003. ISSPIT 2003. Proceedings of the 3rd IEEE International Symposium on, 2003, pp. 696-700.

[140] B. Longworth "A Masterpiece of Deep Learning: GTC Provides Canvas for a Revolutionary Style of Artificial Intelligence," Nvidia, https://blogs.nvidia.com/blog/2016/04/05/artificial-intelligence/, April 5, 2016.

[141] J.-H. Huang "Accelerating AI with GPUs: A New Computing Model", Nvidia, https://blogs.nvidia.com/blog/2016/01/12/accelerating-ai-artificial-intelligence-gpus/, Jan. 12, 2016.