International Journal of Parallel Programming
https://doi.org/10.1007/s10766-021-00713-2

Parallel Computation of Discrete Orthogonal Moment on Block Represented Images Using OpenMP

Iraklis M. Spiliotis[1] · Charalampos Sitaridis[1] · Michael P. Bekakos[1]

Received: 26 September 2020 / Accepted: 31 March 2021

Abstract
Herein, a parallel implementation of Discrete Orthogonal moments on block represented images is investigated. Moments and moment functions have been used widely as features for image analysis and pattern recognition tasks. The main disadvantage of all moment sets, is the high computational cost which is increased as higher-order moments are involved in the computations. In image block representation (IBR) the image is represented by homogeneous areas which are called blocks. The IBR allows moment computation with zero computational error for binary images, low computational error for gray images, low computational complexity, while can achieve high processing rates. The results from parallel implementation on a multicore computer using OpenMP, exhibit significant performance.

Keywords Discrete orthogonal moments · Image block representation · Fast image analysis · Parallel algorithms · OpenMP

1 Introduction and Related Work

Moments and moment functions used widely as features in image analysis and computer vision applications [1–4]. The orthogonal moments use orthogonal polynomials as basis functions, thus they describe a signal with minimal information redundancy as compared to geometric moments and their variations. The continuous

✉ Iraklis M. Spiliotis
spiliot@ee.duth.gr

Charalampos Sitaridis
csitarides@gmail.com

Michael P. Bekakos
mbekakos@ee.duth.gr

[1] Department of Electrical and Computer Engineering, Democritus University of Thrace, 67100 Xanthi, Greece

Published online: 15 April 2021

orthogonal moments have the problem of discretization error. The discrete orthogonal moments are based on discrete orthogonal polynomials, and they have superior image representation ability than other moment sets. The most familiar discrete orthogonal moment (DOM) sets are the Tchebichef [5], the Krawtchouk [6], and the Hahn moments [7, 8]. The reconstruction of an image from a finite number of discrete orthogonal moments [5, 6] indicates the discriminative ability of moments. The Hahn moments considered as a generalization of Tchebichef and Krawtchouk moments. The Tchebichef moments are able to capture the global features of an image while the Krawtchouk moments capture local features. It has been proved that Hahn moments have better image reconstruction quality as compared to the Tchebichef, Krawtchouk, Charlier [9], Meixner [10], and Racah [11] discrete orthogonal moments.

Discrete orthogonal moments proposed as features in several computer vision and pattern recognition tasks [12], as face description [13], 3D object description [14–16], feature extraction from encrypted images [17], image watermarking [18] and improved image classification [19, 20].

A significant drawback of moments is the required high computational cost [21]. Several approaches have been proposed for the reduction of this computational effort. In Image Block Representation (IBR) [22], the binary image is represented as a set of homogeneous rectangulars with edges parallel to image axes, which called blocks. The development of algorithms that operate in blocks instead of image pixels, results in significant computation time savings due to the provided intrinsic parallelism from the IBR. In specific, algorithms for image preprocessing [22], skeletonization [23], thinning [24] and Hough transform [25] have been studied. Also, the real-time computation of geometric moments in block represented binary images has been proposed [26]. An extension of IBR on gray images permits the real-time computation of geometric moments [27, 28] and the fast computation of the Hahn moments [29] to gray images. This latter sequential work is the basis of the parallel work presented in this paper.

Herein, the IBR scheme is used for the parallel computation of DOM of binary and gray images. In the proposed method, the binary image is presented by blocks and the fast computation of DOM in blocks is achieved. In the case of a gray image, the image is decomposed into a set of binary images, the most significant ones are represented by blocks and the DOM are computed fast. The least significant binary images substituted by a constant ideal image called “half-intensity” image, with known DOM values.

Moment computation is not the only example with high required computational cost; images include a significant amount of information and image processing involves complex algorithms. For this reason parallel processing has been used to achieve faster processing rates. The Shared Memory Parallel Machine (SMPM), a multicore shared memory computer which usually uses the OpenMP (Open Multi-Processing) API [30] has been used in image processing and analysis tasks. OpenMP API is also suitable for Intel Xeon Phi coprocessor accelerators [31] and GPU accelerators [32]. Some examples of parallel image and video processing on SMPM are the segmentation of moving objects from background in a video [33], the continuous orthogonal Legendre moment computation [34], the image segmentation

[35] using watershed operation and the extraction of SIFT and SURF features [36]. Recently, an OpenMP parallel implementation of the Image Block Representation has been presented [37].

The rest of the paper is organized as follows: In Sect. 2 the DOM are presented. In Sect. 3, we present the block representation and moment computation method. In Sect. 4 we propose a parallel algorithm for the DM computation using OpenMP, while in Sect. 5 the theoretical analysis of the parallel algorithm is presented. Section 6 presents the experimental results and finally Sect. 7 concludes the work.

2 Discrete Orthogonal Moments

The 2-D Discrete Orthogonal moment of order pq of an image intensity function $f(x,y)$ with size $N_x \times N_y$ is defined as:

$$M_{pq} = \sum_{x=0}^{N_x-1} \sum_{y=0}^{N_y-1} P_p(x, N_x) P_q(y, N_y) f(x, y) \tag{1}$$

where $p = 0, 1, 2, \ldots, N_x - 1$, $q = 0, 1, 2, \ldots, N_y - 1$. The $P_p(x, N_x)$, $P_q(y, N_y)$ are the pth and qth order discrete orthogonal polynomials, which defined by recurrent relations. The polynomials $P_p(x, N_x)$, $P_q(y, N_y)$ are 1D and are identical for a square image where $N_x = N_y$. An image with size $N_x \times N_y$ can be reconstructed from a restricted number of DOM from order (0, 0) up to the order (P,Q) using the inverse moment transform and this indicates the representative power of the moments.

$$\hat{f}(x, y) = \sum_{x=0}^{N_x-1} \sum_{y=0}^{N_y-1} P_p(x, N_x) P_q(y, N_y) M_{pq}, \tag{2}$$

where $\hat{f}(x, y)$ is the reconstructed image. If the number of moments is equal to the number of image pixels, then the reconstructed image is identical to the original image.

It is observed from (1) the high computational complexity for DOM, specifically for a square image with NxN pixels and for moment order from (0, 0) up to (Q,Q), the complexity is $O(Q^2N^2)$. The type of discrete orthogonal polynomial kernel determines which type of DOM set is calclulated. In the sequel the Hahn polynomials, as a representative discrete orthogonal polynomial set are presented.

2.1 Hahn Polynomials

The $h_p^{(\mu,v)}(x, N)$ is the pth order orthogonal Hahn polynomial, defined by the following recursive relation:

$$Ah_p^{(\mu,v)}(x,N) = B\sqrt{\frac{d_{p-1}^2}{d_p^2}}h_{p-1}^{(\mu,v)}(x,N) + C\sqrt{\frac{d_{p-2}^2}{d_p^2}}h_{p-2}^{(\mu,v)}(x,N), \quad p = 2,3,\ldots,N-1 \tag{3}$$

with

$$\begin{aligned} h_0^{(\mu,v)}(x,N) &= \sqrt{\frac{\rho(x)}{d_0^2}} \\ h_1^{(\mu,v)}(x,N) &= \{(N+v-1)(N-1) - (2N+\mu+v-2)x\}\sqrt{\frac{\rho(x)}{d_1^2}} \end{aligned} \tag{4}$$

where $\mu, v\,(\mu > -1,\ v > -1)$ are adjustable parameters and the values $\mu, v = 0$ are suitable for the global feature extraction of the image. The parameters A, B, and C are defined as:

$$\begin{aligned} A &= \frac{p(2N+\mu+v-p)}{(2N+\mu+v-2p+1)(2N+\mu+v-2p)} \\ B &= x - \frac{2(N-1)+v-\mu}{4} - \frac{(\mu^2-v^2)(2N+\mu+v)}{4(2N+\mu+v-2p+2)(2N+\mu+v-2p)} \\ C &= \frac{(N-p+1)(N-p+\mu+1)(N-p+v+1)(N-p+\mu+v+1)}{(2N+\mu+v-2p+2)(2N+\mu+v-2p+1)} \end{aligned} \tag{5}$$

The weighting function $\rho(x)$ can be solved by using the recursive relation concerning x as:

$$\begin{aligned} \rho(x) &= \frac{(N-x)(N-v-x)}{x(x+\mu)}\rho(x-1), \quad x = 1,2,\ldots,N-1 \\ \rho(0) &= \frac{1}{\Gamma(\mu+1)\Gamma(N+v)\Gamma(N-p)} \end{aligned} \tag{6}$$

The discrete orthogonal polynomial with respect to the y axis $P_q(y,N_y)$ is calculated in the same way; for square images $P_q(y,N_y)$ has identical values with $P_p(x,N_x)$.

3 Moment Computation

3.1 Binary Images

In a binary image, the object level pixels have intensity 1 and the background pixels have intensity 0. The pixels with object-level are represented by a set of non-overlapping rectangles with edges parallel to the axes, in such a way that every object pixel belongs to only one rectangle. These formed rectangles are called blocks; this representation is lossless and is called Image Block Representation (IBR). The IBR process, is a fast process without numerical computations and requires one image scan and simple pixel checking operations.

Algorithm 1 Serial Image Block Representation.

Step 1:	Consider each line y of the image f and find the object level intervals in line y.
Step 2:	Compare intervals of line y with blocks of line $y-1$.
Step 3:	If an interval does not match with any block, this is the beginning of a new block.
Step 4:	If a block matches with an interval, the end of the block is in line y.

A binary image represented by blocks is described as $f(x,y) = \{b_i : i = 0, 1, \ldots, k-1\}$, where k is the number of the blocks and b_i is the ith block described by the coordinates of two opposite diagonal angular points, as $b_i = \left(x_{1,b_i}, x_{2,b_i}, y_{1,b_i}, y_{2,b_i}\right)$.

Since the background pixels have intensity 0, only the foreground pixels belong to the k blocks, will take part in the calculation of the moments. The foreground pixels and Thus a DOM of order pq of a binary image can be defined as:

$$M_{pq} = \sum_{i=0}^{k-1} \sum_{x=0}^{N_x-1} \sum_{y=0}^{N_y-1} P_p(x, N_x) P_q(y, N_y) \tag{7}$$

Exploiting the rectangular form of the block with edges parallel to the image axes, the double sum of discrete orthogonal polynomials can be rewritten as the product of two separate sums. Each sum contains the polynomial terms for the horizontal and vertical axis of the block, respectively.

$$M_{pq}^{b} = \sum_{x=x_{1,b}}^{x_{2,b}} P_p(x, N_x) \sum_{y=y_{1,b}}^{y_{2,b}} P_q(y, N_y) \tag{8}$$

and the moments of the whole image are

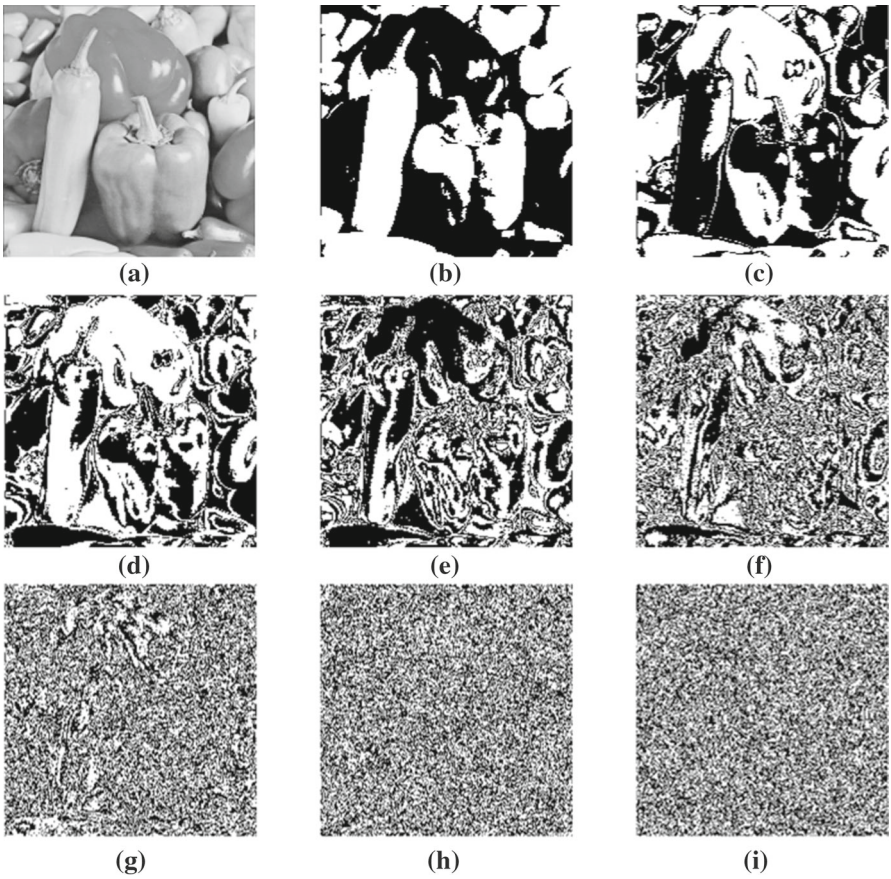

Fig. 1 Decomposition of the gray image peppers with 256 gray levels into 8 binary images. **a** The original gray image. **b–i** The binary images p_7 at **b** derived from the most significant bits, and p_0 at **i** derived from the least significant bits

$$M_{pq} = \sum_{i=0}^{k-1} \sum_{x=0}^{N_x-1} P_p(x, N_x) \sum_{y=0}^{N_y-1} P_q(y, N_y) \tag{9}$$

The computational complexity using (9) $O(N)$, instead of $O(N^2)$ when using (1).

3.2 Gray and Color Images

A gray image has an intensity function $g(x,y)$ and 2^n gray levels. This gray image can be decomposed into n binary images. Each of these binary images, is a bit plane p_i of the original gray image derived from the bits of the same significance of the values of the corresponding pixel of the gray image. The bit-plane p_{n-1} is consisted of the most significant bits (MSB), while the bit plane p_0 consisted of the least

significant bits (LSB) of the gray image pixel values. The relation between the gray image $g(x, y)$ and the n bit planes is

$$g(x,y) = 2^{n-1}p_{n-1}(x,y) + \cdots + 2^1 p_1(x,y) + 2^0 p_0(x,y) \tag{10}$$

An example of the decomposition of an image is illustrated in Fig. 1, where the initial image of Fig. 1a with 256 gray levels, is decomposed to the 8 corresponding bit planes of Fig. 1b–i. It is easy to notice that the binary images of lower order are noisy. In particular, the reduction of the computational cost of moment calculation is based on this observation and in the fact that the n binary images resulting from the decomposition of the gray image can be represented by blocks.

Taking into account Eq. (10), the DOM of order pq of the gray image g, is calculated as:

$$\begin{aligned} M_{pq} &= \sum_{x=0}^{N_x-1}\sum_{y=0}^{N_y-1} P_p(x,N_x)P_q(y,N_y)g(x,y) \\ &= \sum_{x=0}^{N_x-1}\sum_{y=0}^{N_y-1} P_p(x,N_x)P_q(y,N_y)\left[2^{n-1}p_{n-1}(x,y) + \cdots + 2^1 p_1(x,y) + 2^0 p_0(x,y)\right] \\ &= \left(2^{n-1}Mp_{(n-1)pq} + \cdots + 2^1 Mp_{1pq} + 2^0 Mp_{0pq}\right) = \sum_{i=0}^{n-1} 2^i Mp_{ipq} \end{aligned} \tag{11}$$

where $p_{n-1}(x,y), \ldots, p_1(x,y), p_0(x,y)$ are the bit planes that compose the gray image $g(x, y)$, with $Mp_{(n-1)pq}, \ldots, Mp_{1pq}, Mp_{0pq}$ their DOM of order pq, which are calculated fast by using IBR and Eq. (9).

As implied by Eq. (10), due to the factors 2^i, the bit planes p_i do not contribute equally to the gray image and the order of each bit plane indicates its significance. Moreover from Fig. 1, it is observed that the lower order bit planes are quite noisy with continuous black-and-white transitions and are similar to a chessboard image or simply with a "half intensity" image h with constant intensity 1/2.

It has been shown [29] that the moment values of an image with intensity 1/2 are the half of the moment values of an image with intensity 1. Considering the half-intensity image $h = 1/2,\ \forall x, y$ as one block, the substitution of the $(n\text{-}m)$ least significant bit planes with the half intensity image $h(x, y)$ results to the DOM $M_{m,pq}$, by replacing

$$M_{m,pq} = \sum_{i=n-m}^{n-1} 2^i Mp_{ipq} + \sum_{j=0}^{n-1-m} 2^j Mh_{pq} = \sum_{i=n-m}^{n-1} 2^i Mp_{ipq} + Mh_{pq}\sum_{j=0}^{n-1-m} 2^j \tag{12}$$

In the above Eq. (12) the DOM Mp_{ipq} of the m higher-order bit planes are computed fast using IBR and Eq. (9). Considering that a half-intensity plane is equivalent with a block with image size and intensity ½, the DOM Mh_{ipq} of the half-intensity planes they are computed fast. Additional computational time gains can be

obtained if the DOM of all the half-intensity planes are pre-calculated, stored, and used in Eq. (12).

The replacement of the (n-m) lower order bit planes with half-intensity images results in an artificial image $\hat{g}$, which is an approximation of the original input image g. To evaluate the quality of the approximated gray image both the subjective observation from the human visual system and an objective metric were used. As shown in [29], it is adequate to use 3 or 4 real bit planes. A suitable image quality metric is the Normalized Image Reconstruction Error (NIRE) [38, 39], which is the normalized square error between the image functions g and $\hat{g}$, is defined as

$$NIRE(g, \hat{g}) = \frac{\sum_x \sum_y [g(x,y) - \hat{g}(x,y)]^2}{\sum_x \sum_y g^2(x,y)} \tag{13}$$

In the Sect. 6 evaluations of the reconstruction quality are given.

In the case of *color images*, they are decomposed in the three color components Red, Green, Blue and each component is handled as a gray image using the procedure described in this Section.

4 Parallel DOM Computation Using OpenMP

The Algorithm 2 describes the parallel computation of the Discrete Orthogonal moments.

Algorithm 2 Parallel DOM computation.

Step 1.	In the case of a gray image execute the image decomposition task.
	In the case of a binary image go to step 2.
Step 2.	For all used real bit planes of a gray image or for the binary image.
Step 3.	Parallel Image Block Representation of the bit plane.
Step 4.	Parallel DOM computation of the bit plane.
Step 5.	Add weighted bit plane moments to the image DOM.
Step 6.	Next bit plane.
Step 7.	In the case of a gray image add moments of half-intensity images to DOMa of gray image.

In the sequel the significant steps of the above Algorithm 2 are clarified.

4.1 Parallel Decomposition of the Gray Image

Usually the number of gray levels is 256, thus the corresponding bit planes after the decomposition are 8. As already mentioned, it is adequate to extract 3 or 4 real bit planes, since the 5 or 4 lower bit planes are replaced by-half intensity images.

In OpenMP the static, dynamic, and guided scheduling mechanisms and chunk size can be specified for loop partitioning. Static schedule divides the loop into

equal-sized chunks among the threads. Dynamic schedule uses a work pool with the loop iterations and assigns chunk sized bundles of iterations to threads, when a thread finished retrieves the next chunk sized bundle from the work pool. The guided schedule is similar to dynamic, but the chunk size begins from a large value and decreases to handle efficiently the load imbalance among the threads. The specified chunk size in a guided schedule is the minimum value to use. In static scheduling the default chunk size is the quotient of the number of iterations divided by the number of threads, while in dynamic and guided scheduling the default chunk size is 1.

The rows of the gray image are partitioned among the threads using the guided scheduling. In every thread a pixel by pixel processing is applied; each thread receives a byte corresponding to the pixel value of the gray image, extracts the bits, and sets the values of the corresponding pixel on the bit planes. The required execution time of the gray image decomposition is negligible in comparison to IBR and moment computation.

4.2 Parallel Implementation of IBR

The parallel OpenMP IBR algorithm (PIBR) has been developed and presented recently [37], a short description is presented here. The PIBR Algorithm is constituted from two parts:

Part 1: Interval Extraction The input of PIBR algorithm is an image in a 2D array form. The L image rows are partitioned equally among the P threads, each thread t is assigned a bundle of L/P image rows using a static partitioning scheme; the static partitioning ensures that each thread t is assigned the following image

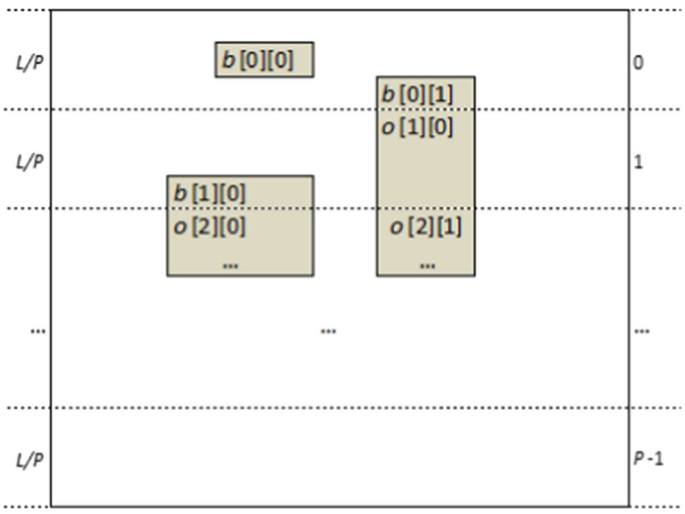

Fig. 2 The L image rows are partitioned among the P threads. The image has three blocks, $b[0][0]$ which lies entirely at the rows of thread 0, $b[0][1]$ starts at thread 0 and its continuation at thread 1 is initially registered as orphan $o[1][0]$; $b[1][0]$ starts at thread 1 and its part at thread 2 is initially registered as orphan $o[2][0]$

rows [t*L/P, (t + 1)*L/P − 1]. Each thread t extracts the object level intervals in the L/P image rows.

Part 2: Interval Matching Each thread matches the extracted intervals among consecutive image rows and creates the blocks. There are two types of blocks, the regular blocks which are denoted as b and orphan blocks which are denoted as o. The orphan blocks are temporary ones, they start at the image rows of a previous thread, and finally are merged into a regular block.

The output of PIBR algorithm is the array of blocks b[][], the first index of the array is the thread id and the second index of the array is the block id with regards to thread. Since the number of extracted blocks per thread is different, the rows of the array have different lengths. The array b[][] uses non-contiguous memory and in cases that a block has to be accessed isolated from a previous block, then the access time is $O(P)$. Since P is the number of threads, it has relatively small values and does not introduce any drawback. Figure 2 demonstrates the PIBR algorithm.

4.3 Parallel Computation of DOM

The moment computation is fully parallelizable, since there are no data race conditions and no communication among the processors is required; this is achieved using loop partitioning.

As implied by Eq. (12) the procedure for a gray image has two parts, one for the higher order real bitplanes and one for the lower order half-intensity planes and is demonstrated in Fig. 3.

The processing of the real bitplanes is implemented in a sequential manner. The blocks of each bitplane are partitioned among the threads, using the *guided* OpenMP

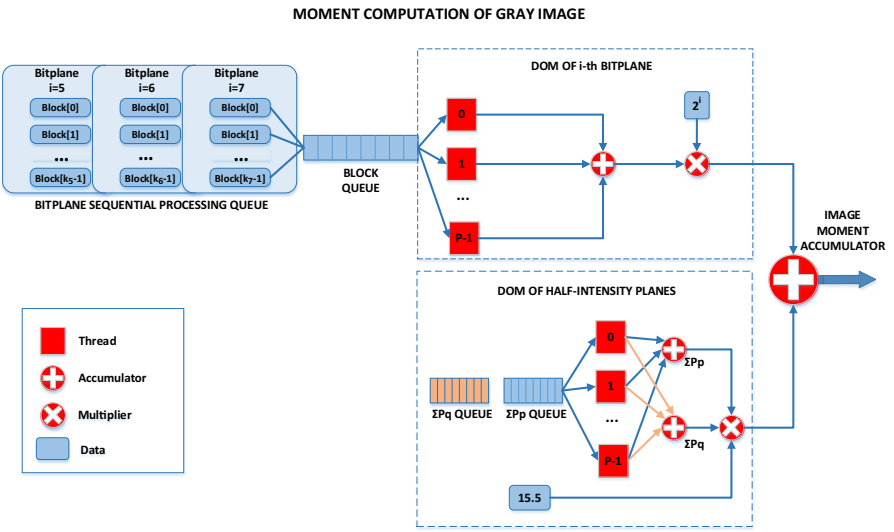

Fig. 3 The procedure for the parallel computation of moments of a gray image with 3 real and 5 half-intensity bitplanes

schedule to utilize better the available threads. The computation of the moments of a block is executed according to Eq. (8). It is obvious by Eq. (8) that the complexity depends on the width and length of the block and is different for each block. Each thread receives the four coordinates of a block, computes the moments of the block using (8), and accumulates moment values in thread's private variables. The thread's local moments are added by the master thread using reduction. Finally, the moments of the bit plane are multiplied by the weight factor 2^i, where i is the index of the bit plane and accumulated to the moments of the gray image.

The above procedure is also applied for computation of DOM of binary images.

In the second part the moments of the half-intensity-planes are calculated. Since all planes consisted of one block with intensity ½, their DOM have the same values which are computed according to (8). Since the size of the block is as large as the image (N_x, N_y), the summations of polynomias in (8) are executed in parallel by all threads using reductions. In the sequel the DOM is multiplied by ½ because of the intensity level and by the the sum of the weight factors 2^i, where i is the index of every half-intensity plane and then added to the moments of the gray image. For example in Fig. 3 the DOM of the block is multiplied by $(2^4 + 2^1 + \cdots + 2^0)/2 = 15.5$. As already discussed the moment values of half-intensity planes may be precalculated.

For color images the procedure is repeated three times, each time the DOM of the gray image, that corresponds to a color component, is calculated. Finally, the summation of these moment values are the moments of the color image.

5 Theoretical Analysis of the Parallel Implementation

The time complexity of a parallel algorithm is the summation of the execution time of the processors and the parallel overhead t_o [40]. For applications with enough available parallelism and without data dependencies, the parallel overhead usually consisted of three different components.

1. The creation of additional instructions inserted by the compiler to fork and join the threads.
2. The idling time of the processors due to imbalanced workload.
3. The memory wall [41] caused by the increasing discrepancy in processor and memory performance. The granularity [42] impacts on the memory wall problem, as high rates of memory reads/writes trigger the cache coherence effect. When the memory accesses are distributed in-depth in different cache lines [43] the whole effect of memory wall problem is decreased.

The parallel overhead depends on the number of processors P and the problem size w which is may be considered as the total number of basic operations. The parallel overhead t_o acts like a serial fraction e of work that existed in parallel execution.

The proposed algorithm consisted of the different parts presented in Sect. 4, which are the parallel gray image decomposition to the bit planes, the parallel IBR, and the parallel computation of DOM on block represented images. The

decomposition of the image to the bitplanes is required only for grayscale images, in the case of binary images this step is meaningless. The time complexity of image decomposition is negligible in comparison with IBR and moment computation, for this reason theoretical analysis only for the other two parts is investigated.

5.1 Estimation of the Execution Time of PIBR Algorithm

The estimation of the execution time of the PIBR algorithm has been determined in a previous work [37] as

$$\hat{t}_{PIBR}(P) = \frac{t_E + t_M}{P} + t_E \cdot e_{ZE}(P) + t_M \cdot e_{ZM}(P) \tag{14}$$

where t_E, t_M are the execution times of the serial interval extraction, and the serial interval matching, e_{ZE}, e_{ZM} are the experimentally determined serial fractions of the parallel interval extraction and interval matching for the zero image of the same size as the input image. The Karp–Flatt metric [40, 44] for the experimentally determined serial fraction, defined as:

$$e(P) = \frac{1/S(P) - 1/P}{1 - 1/P} \tag{15}$$

and used for the determination of $e_{ZE}(P)$, $e_{ZM}(P)$. Assuming the same serial fraction for any other input image with the same size, (14) is derived.

5.2 Estimation of the Execution Time of DOM Computation

According to Eq. (8), the execution of the moment computation of one block requires the sums of the polynomial values over the edges of the block and the product of the two sums. Considering a block represented binary image with consisted of k blocks, the total number A of additions and M of multiplications are

$$\begin{aligned} A &= \sum_{i=0}^{k-1} (x_{2,b_i} - x_{1,b_i} + y_{2,b_i} - y_{1,b_i} + 2) \\ M &= k \end{aligned} \tag{16}$$

The above Eq. (16) defines the problem size w of moment computation, which is the number of basic operations. Equivalently the problem size expressed by the serial execution time

$$t_S = t_A A + t_M k = t_A \sum_{i=0}^{k-1} (x_{2,b_i} - x_{1,b_i} + y_{2,b_i} - y_{1,b_i} + 2) + t_M k \tag{17}$$

where t_A, t_M the execution time of addition and multiplication. The processing time for parallel computation using P threads is

$$t_p(P) = \frac{t_s}{P} + t_o(P) = \frac{t_A \sum_{i=0}^{k-1} (x_{2,b_i} - x_{1,b_i} + y_{2,b_i} - y_{1,b_i} + 2) + t_M k}{P} + t_o(P) \quad (18)$$

where $t_o(P)$ is the parallel overhead time. The moment computation is fully parallelizable, since no data race conditions exist and no communication among the processors is required. The parallel overhead $t_o(P)$ could not be derived analytically, for its determination the procedure described in Sect. 5.1 and [37] is used.

Profiling the moment computation algorithm for the blocks of an input image, and for serial and parallel execution with different number of cores used, the execution times are obtained. This input image is called reference image R and used for the determination of parallel overhead of all the other input images. Considering a serial fraction of the problem e_R, then the Karp–Flatt metric is used to experimentally determine the serial fraction e_R for reference image R using Eq. (15).

For other input images, it is assumed that the serial fraction is the same with the reference image of the same size. Thus the estimated parallel DOM computation (HM) execution time is

$$\hat{t}_{HM}(P) = \frac{t_{HMs}}{P} + e_R \cdot t_{HMs} \quad (19)$$

The assumption that the serial fraction of DOM computation, is of similar value among the reference and the input image, is quite reasonable and takes into account the creation of the parallel region, the number of cores used and also the size of the image which is related to the size of blocks.

6 Experimental Results

For the experimental evaluation, the Greek Research & Technology Network (GRNET) platform *National HPC facility ARIS* was utilized, consisting of DELL PowerEdge R820 nodes, with four Intel(R) Xeon(R) CPUs E5-4650v2, based on Sandy Bridge-EP microarchitecture, with nominal frequency 2.4 GHz, with 40 cores (10 cores/CPU) and 512 GB RAM. The operating system is Centos 6.7 Linux. All the programs were implemented in C using the OpenMP API and were compiled using the Intel compiler icc ver. 15.0.3.

For the performance evaluation of the parallel algorithm the DIVerse 2K (Div2K) dataset [45, 46], which contains 1000 high quality color images of 2K resolution at least in one of vertical or horizontal axis has been utilized. The images use 3 bytes/pixel, one byte for each of the Red, Green and Blue color components. The images of the Div2K dataset are converted to gray using all the three color components to obtain a gray images dataset. Also the gray images are converted to binary using Otsu method [47] for the selection of the binarization threshold per image. The binary, the gray and the color images dataset were used in the experiments. In Fig. 4, some original color images and the corresponding gray and binary images are presented.

Fig. 4 The three first images of the Div2K dataset and their corresponding gray and binary images

Also, a binary and a gray image datasets with small number of images, are used in order to test the scalability of the proposed method. These later datasets contain square images which are available in a variety of sizes.

In pattern recognition applications a small number of moment values can describe the patterns. For the Div2K dataset two moment sets are calculated, one moment set is from moment order (0, 0) up to the moment order (64, 64) and the second moment set is from moment order (0, 0) up to the moment order (128, 128).

Table 1 The average time complexities, speedup and efficiency for DOM computation of the binary images derived from Div2K dataset

	IBR (ms)	DOM order (0, 0)–(64, 64)				DOM order (0, 0)–(128, 128)			
		BM (s)	Total (s)	S_R	E_R	BM (s)	Total (s)	S_R	E_R
t_s	7.986	0.307	0.315			1.073	1.081		
$t_p(1)$	8.138	0.311	0.319	1.00	1.00	1.093	1.101	1.00	1.00
$t_p(2)$	4.618	0.163	0.168	1.90	0.95	0.564	0.569	1.94	0.97
$t_p(4)$	2.512	0.084	0.086	3.70	0.93	0.286	0.289	3.81	0.95
$t_p(8)$	1.624	0.044	0.045	7.02	0.88	0.148	0.150	7.35	0.92
$t_p(16)$	0.840	0.023	0.024	13.11	0.82	0.077	0.078	14.10	0.88
$t_p(32)$	0.586	0.014	0.014	22.03	0.69	0.042	0.042	26.11	0.82
$t_p(40)$	0.523	0.013	0.014	23.17	0.58	0.040	0.041	27.06	0.68

For the smaller datasets with square images of size $L \times L$, one moment set order (0, 0) up to the order ($L/16 \times L/16$) is calculated.

The computation times were measured from the execution of the proposed algorithms, excluding image reading from disk, etc. All-time complexities are the average of repetitive executions. The presented experimental results are for the computation of Hahn moments. However the computation of other moment sets as Tchebishef, Krawtchouk, Racah e.t.c. has similar computation times as a number of experiments shown.

6.1 Experimental Results for Binary Images

In Table 1, the average time complexities and the achieved speedup and efficiency values for the Hahn moment computation on the images of the binarized version of the Div2K dataset for serial and parallel executions are demonstrated, where IBR is the time for block representation, BM is the time for moment computation using the blocks and total is the overall time. In Fig. 5 the relative speedup values achieved for the parallel Hahn moment computation for different moment orders are demonstrated. From the results it is observed that the proposed method achieves significant speedup values.

6.2 Experimental Results for Gray Images

As discussed in Sect. 3.2, it is adequate to represent a gray image g with an artificial image $\hat{g}_m$ constructed from m of higher order bitplanes of g and (8-m) half-intensity images. It has been observed that preserving only the first three or four bit planes, while replacing the rest with half intensity images the resulted artificial images $\hat{g}_3$ and $\hat{g}_4$ are acceptable using the subjective criterion of the human visual system and the objective NIRE error metric. An example of this representation is demonstrated in Fig. 6 and in Table 2 the NIRE metric of the reconstructed images is presented.

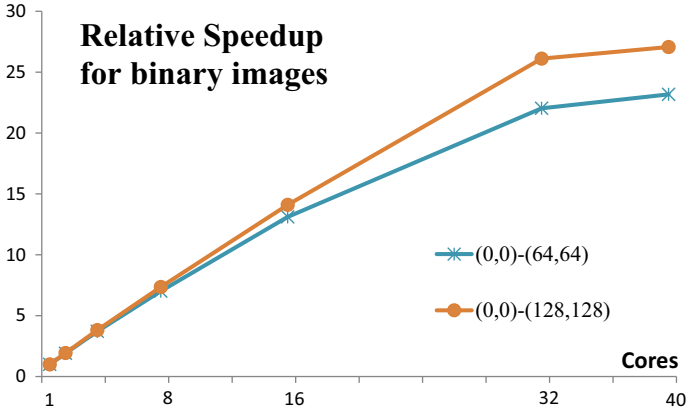

Fig. 5 The relative speedup for DOM computation of the binary images derived from Div2K dataset

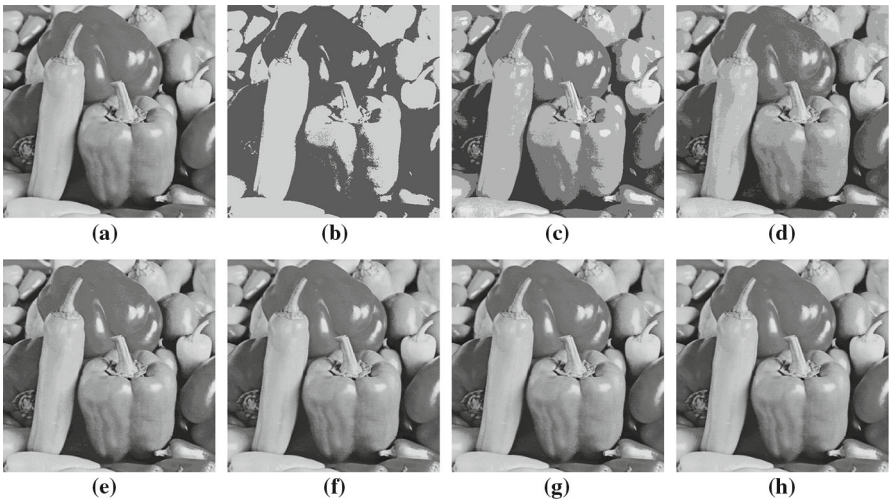

Fig. 6 **a** The input gray image and **b–h** the reconstructed images $\hat{g}_1$, $\hat{g}_2$, …, $\hat{g}_7$

Table 2 The NIRE error metric between the input gray image of Fig. 6a and the corresponding reconstructed images $\hat{g}_k$ constructed from m most significant image bitplanes and (8-m) half-intensity planes

Image	$\hat{g}_1$	$\hat{g}_2$	$\hat{g}_3$	$\hat{g}_4$	$\hat{g}_5$	$\hat{g}_6$	$\hat{g}_7$
NIRE	0.261	0.135	0.069	0.037	0.018	0.009	0.005

Table 3 The average time complexities, speedup and efficiency for DOM computation of the $\hat{g}_3$ gray images derived from Div2K dataset

	$\hat{g}_3$ images		DOM order (0, 0)–(64, 64)				DOM order (0, 0)–(128, 128)			
	DEC3 (ms)	IBR3 (ms)	BM (s)	Total (s)	S_R	E_R	BM (s)	Total (s)	S_R	E_R
t_s	6.11	30.81	2.04	2.08			7.87	7.91		
$t_p(1)$	21.11	36.24	2.17	2.23	1.00	1.00	8.48	8.54	1.00	1.00
$t_p(2)$	10.70	26.36	1.16	1.20	1.86	0.93	4.51	4.54	1.88	0.94
$t_p(4)$	5.76	20.68	0.58	0.61	3.66	0.92	2.34	2.37	3.61	0.90
$t_p(8)$	2.67	15.06	0.30	0.32	7.00	0.88	1.23	1.24	6.87	0.86
$t_p(16)$	1.47	11.51	0.16	0.17	13.16	0.82	0.67	0.68	12.53	0.78
$t_p(32)$	0.96	7.95	0.10	0.11	20.45	0.64	0.41	0.41	20.59	0.64
$t_p(40)$	1.00	7.36	0.09	0.10	22.64	0.57	0.35	0.36	23.82	0.60

Table 4 The average time complexities, speedup and efficiency for DOM computation of the $\hat{g}_4$ gray images derived from Div2K dataset

	$\hat{g}_4$ images		DOM order (0, 0)–(64, 64)				DOM order (0, 0)–(128, 128)			
	DEC4 (ms)	IBR4 (ms)	BM (s)	Total (s)	S_R	E_R	BM (s)	Total (s)	S_R	E_R
t_s	9.35	57.16	3.93	4.00			14.15	14.22		
$t_p(1)$	27.14	62.14	4.13	4.22	1.00	1.00	14.37	14.46	1.00	1.00
$t_p(2)$	12.82	45.09	2.16	2.22	1.91	0.95	7.61	7.67	1.88	0.94
$t_p(4)$	7.03	37.19	1.11	1.16	3.65	0.91	4.01	4.06	3.57	0.89
$t_p(8)$	3.45	27.16	0.56	0.59	7.12	0.89	2.07	2.10	6.88	0.86
$t_p(16)$	1.96	21.18	0.29	0.32	13.29	0.83	1.11	1.13	12.76	0.80
$t_p(32)$	1.49	17.42	0.22	0.24	17.67	0.55	0.72	0.74	19.57	0.61
$t_p(40)$	1.43	15.93	0.21	0.23	18.41	0.46	0.65	0.67	21.67	0.54

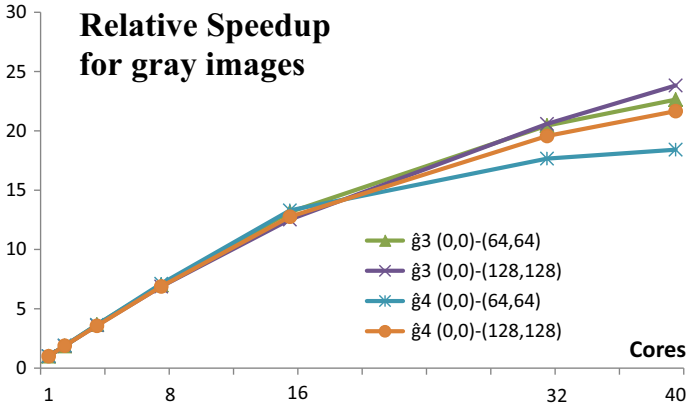

Fig. 7 The relative speedup for DOM computation of the gray images derived from Div2K dataset

In Table 3, the average time complexities and the achieved speedup and efficiency values for the Hahn moment computation on the $\hat{g}_3$ gray images derived from Div2K dataset for serial and parallel executions are demonstrated, while in Table 4 the same results for $\hat{g}_4$ gray images of Div2K dataset, where DECm is the time for the decomposition of the gray image in m bitplanes, IBRm is the time for the block representation of m bitplanes, BM is the time for the moment computation on the real and half-intensity planes using the blocks and total is the overall time. In Fig. 7 the relative speedup values achieved for the parallel Hahn moment computation for different moment orders are demonstrated. It is observed that the proposed method achieves high speedup values.

A significant observation concerns the granularity of the parallelism [42, 48]. The achieved speedup values are greater for binary images as compared to gray images.

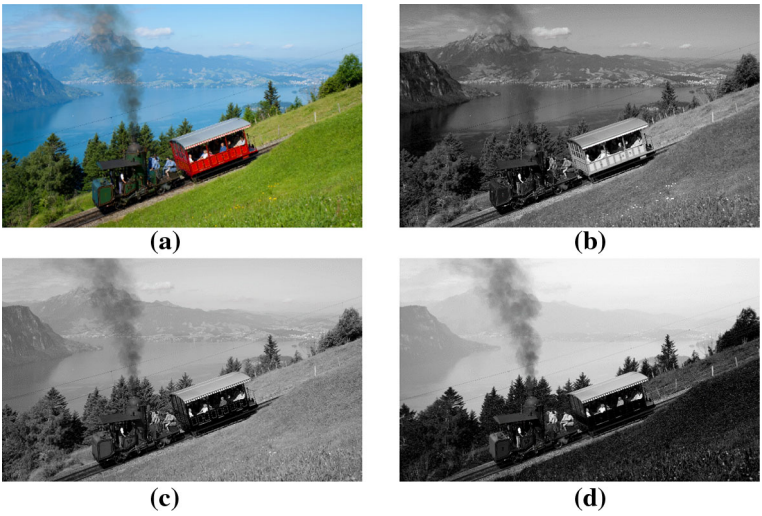

Fig. 8 **a** A color image of the Div2K dataset, **b** the red, **c** the green, **d** and the blue color components of **a** (Color figure online)

In gray images and especially in lower significant bit planes there is a huge number of small blocks, that correspond to a huge number of tasks with small problem size; this situation leads to an increased memory access time that is comparable to processing time and described as fine-grained parallelism. In binary images coarse-grain parallelism appeared, since there is a lower number of tasks with increased problem size.

The granularity issue is also observable in gray image representations $\hat{g}_3$ and $\hat{g}_4$, the first representation with 3 real bit planes and 5 half-intensity images has greater speedup values than the second image representation with 4 real bit planes and 4 half-intensity images.

6.3 Experimental Results for Color Images

For the representation of color images a suitable color model which usually used is the RGB color model; also this is color model in the Div2K dataset. An RGB image consists of the Red, Green an Blue color components, where each component is a gray image. Figure 8 demonstrates a color image from the Div2K dataset and its

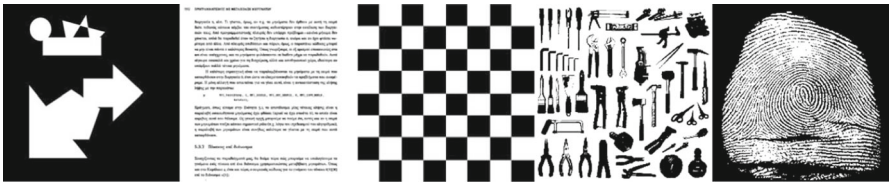

Fig. 9 A set of test binary images: shapes, page, chessboard, tools and fingerprint

Fig. 10 A set of test gray images: island, mountain, drone, horses and MRI

Red, Green and Blue components. For the DOM computation the input color image is extracted to the color components, in the sequel each one of these gray images is handled using the described process for gray images and finally their moment values added in order to get the color image's moment values. As expected, the experimental average time complexities for the color images of the Div2K dataset are three times the time complexities of the Tables 3 and 4 when using $\hat{g}_3$ and $\hat{g}_4$ images respectively; the speedup and efficiency metrics are identical with the corresponding metrics of the gray images of Sect. 6.2.

6.4 Scalability of the Algorithm

The scalability of a parallel algorithm is the ability to increase performance as the number of the processors increases. A scalable system should increase speedup in such a rate that the efficiency is maintained as the number of processors increases. However, the parallel overhead increases as the number of processors increases and the efficiency decreases as the number of processors increases. Increasing the problem size is a way to maintain efficiency. In order to experimentally verify the scalability of the proposed algorithm two image sets are utilized. The first set contains square binary images, the second set contains square gray images and each image is available in size of 1K × 1K, 2K × 2K, 4K × 4K and 8K × 8K pixels.

In Figs. 9 and 10 the binary and the gray image sets are demonstrated. Table 5 presents the average time complexities for Hahn moment computation and the metrics of relative speedup and efficiency for the binary images of Fig. 9. The time complexities of Table 5, are the average of execution time of same sized images and include the serial computation using images represented as 2D array, the serial and the parallel computation for a number of processor cores using block represented images. In order to improve the readability of the results, the time complexities of Table 5 are the total including IBR and computation of moments using blocks, also there is one moment set from order (0, 0) up to the order ($L/16 \times L/16$), where $L \times L$ is the image size.

Table 6 presents the time complexities for Hahn moment computation and the metrics of relative speedup and efficiency for the gray images of Fig. 10. The time complexities of Table 6 are averages of same sized images and include the image decomposition, the block representation and the moment calculation using blocks.

For both binary and gray image cases, the proposed parallel computation is scalable, since increasing both the number of CPU cores and the problem size, the

Table 5 The average time complexities for DOM computation for the binary images of Fig. 9 of different sizes, for serial execution using images represented as 2D array and serial and parallel execution using block represented images

Cores	Image size 1 K × 1 K DOM (0, 0)–(64, 64)			Image size 2 K × 2 K DOM (0, 0)–(128, 128)			Image size 4 K × 4 K DOM (0, 0)–(256, 256)			Image size 8 K × 8 K DOM (0, 0)–(512, 512)		
	T (s)	S_R	E_R	T (s)	S_R	E_R	T (s)	S_R	E_R	T (s)	S_R	E_R
t_{s2D}	18.099			284.03			4567.25			71,804.32		
t_s	0.18			0.78			2.39			8.72		
$t_p(1)$	0.18	1.00	1.00	0.80	1.00	1.00	2.48	1.00	1.00	9.43	1.00	1.00
$t_p(2)$	0.09	1.93	0.97	0.41	1.95	0.97	1.27	1.95	0.97	4.71	2.00	1.00
$t_p(4)$	0.05	3.48	0.87	0.20	3.94	0.99	0.63	3.91	0.98	2.30	4.10	1.02
$t_p(8)$	0.03	6.40	0.80	0.11	7.49	0.94	0.32	7.65	0.96	1.19	7.94	0.99
$t_p(16)$	0.01	12.47	0.78	0.06	13.71	0.86	0.19	13.39	0.84	0.60	15.67	0.98
$t_p(32)$	0.01	17.58	0.55	0.04	21.53	0.67	0.11	22.66	0.71	0.32	29.39	0.92
$t_p(40)$	0.01	18.79	0.47	0.03	23.77	0.59	0.08	29.59	0.74	0.33	28.64	0.72

efficiency values are maintained. The problem size is related to the image size, the image density, and the maximum moment order.

It is worth being noticed that considering the serial 2D array moment computation versus the parallel block moment computation using 40 cores, the achieved speedup $t_{s2D}/t_p(40)$ reach very large values, for example for 1K × 1K images exceeds 1800, for 2K × 2K images exceeds 8000, for 4K × 4K images exceeds 50,000 and for 8K × 8K exceeds 200,000 as resulted from time complexities of Table 5.

7 Conclusion

In this paper a parallel OpenMP computation method of DOM in binary and grayscale images is presented. The proposed method is a parallelization of a sequential method which is based on the decomposition of the input image to the corresponding bit planes and the representation of binary images with blocks. The lower order bit planes can be substituted by a half-intensity image with moment values equal to the half of full intensity image. The image block representation creates an intrinsic parallelism which results in the acceleration of the sequential computation of the DOM on a sequential machine.

From the experimental evaluation it is concluded that the presented method achieves significant reduction of the required computation time and allows the fast parallel computation of discrete orthogonal moments on block represented binary, gray and color images. The parallelization using OpenMP API, achieves significant speedup and efficiency values and permits high processing rates. Also the proposed method for parallel DOM computation is scalable as evaluated from experimental results.

Table 6 The average time complexities for DOM computation for the gray images of Fig. 10 of different sizes, for serial and parallel execution using block represented images

	Image size 1 K × 1 K DOM (0, 0)–(64, 64)			Image size 2 K × 2 K DOM (0, 0)–(128, 128)			Image size 4 K × 4 K DOM (0, 0)–(256, 256)			Image size 8 K × 8 K DOM (0, 0)–(512, 512)		
	T (s)	S_R	E_R	T (s)	S_R	E_R	T (s)	S_R	E_R	T (s)	S_R	E_R
t_s	0.64			5.19			45.01			365.22		
$t_p(1)$	0.91	1.00	1.00	7.62	1.00	1.00	59.49	1.00	1.00	427.55	1.00	1.00
$t_p(2)$	0.51	1.78	0.89	4.19	1.82	0.91	31.84	1.87	0.93	229.39	1.86	0.93
$t_p(4)$	0.27	3.45	0.86	2.15	3.54	0.89	16.06	3.70	0.93	115.85	3.69	0.92
$t_p(8)$	0.14	6.50	0.81	1.10	6.91	0.86	8.36	7.11	0.89	59.40	7.20	0.90
$t_p(16)$	0.08	11.70	0.73	0.64	11.99	0.75	4.74	12.54	0.78	30.85	13.86	0.87
$t_p(32)$	0.06	14.39	0.45	0.37	20.54	0.64	2.82	21.13	0.66	17.40	24.57	0.77
$t_p(40)$	0.04	20.65	0.52	0.33	23.01	0.58	2.22	26.85	0.67	15.40	27.76	0.69

As already discussed in the Introduction, moments are popular features in pattern recognition tasks. Typical pattern recognition systems, usualy consisted of an optional image preprocessing unit, the feature extraction unit and the classification unit. The bottleneck of these systems when use moments, is the feature extraction unit, since the preprocessing and classifier units are assigned with not computationally intensive processes. The acceleration of moment computation creates a similar speedup value in the processing rate of the whole system.

The parallel implementation of the IBR algorithm and the DOM computation, on Distributed Memory Parallel Machines using MPI, on GPGPUs and on FPGAs are interesting directios of our current research.

Acknowledgements This work was supported by computational time Granted from the Greek Research & Technology Network (GRNET) in the National HPC facility - ARIS - under project ID PA170601-PIBR.

References

1. Hu, M.K.: Visual pattern recognition by moment invariants. IRE Trans. Inf. Theory **8**, 179–187 (1962)
2. Teague, M.R.: Image analysis via the general theory of moments. J. Opt. Soc. Am. **70**, 920–930 (1980)
3. Teh, C.-H., Chin, R.T.: On image analysis by the method of moments. IEEE Trans. Pattern Anal. Mach. Intell. **10**, 496–513 (1988)
4. Flusser, J., Suk, T.: Rotation moment invariants for recognition of symmetric objects. IEEE Trans. Image Process. **15**, 3784–3790 (2006)
5. Mukundan, R.: Image analysis by Tchebichef moments. IEEE Trans. Image Process. **10**, 1357–1364 (2001)
6. Yap, P.T., et al.: Image analysis by Krawtchouk moments. IEEE Trans. Image Process. **12**, 1367–1377 (2003)
7. Yap, P.T., et al.: Image analysis using Hahn moments. IEEE Trans. PAMI **29**, 2057–2062 (2007)

8. Zhou, J., et al.: Image analysis by discrete orthogonal Hahn moments. In: Image Analysis and Recognition. ICIAR 2005, Lecture Notes in Computer Science, vol. 3656. Springer, Berlin, Heidelberg (2005)
9. Karmouni, H., et al.: Fast 3D image reconstruction by cuboids and 3D Charlier's moments. J. Real-Time Image Process. **17**, 1–17 (2020)
10. Jahid, T., et al.: Image analysis by Meixner moments and a digital filter. Multimed. Tools Appl. **77**, 19811–19831 (2018)
11. Wu, Y., Liao, S.: Image reconstruction from discrete orthogonal Racah moments. In: IEEE Canadian Conference on Electrical and Computer Engineering (CCECE) (2016)
12. Flusser, J., Zitová, B., Suk, T.: Moments and Moment Invariants in Pattern Recognition. Wiley (2009)
13. Akhmedova, F., Liao, S.: Face recognition using discrete orthogonal Hahn moments. In: International Journal of Computer, Electrical, Automation, Control and Information Engineering, vol. 9 (2015)
14. Mesbah, A., et al.: Robust reconstruction and generalized dual Hahn moments invariants extraction for 3D images. 3D Res. **8**, 1, Article 113 (2017)
15. El Mallahi, M., et al.: Radial Hahn moment invariants for 2D and 3D image recognition. Int. J. Autom. Comput. **15**(3), 277–289 (2018)
16. Mesbah, A., et al.: Fast and efficient computation of three-dimensional Hahn moments. J. Electron. Imaging **25**(6), 061621 (2016)
17. Yang, T., et al.: Image feature extraction in encrypted domain with privacy-preserving Hahn moments. IEEE Access **6**, 47521–47534 (2018)
18. Ahmad, S., Lu, Z.-M.: Geometric distortions-invariant digital watermarking using scale-invariant feature transform and discrete orthogonal image moments. In: Digital Rights Management: Concepts, Methodologies, Tools, and Applications (2013). https://doi.org/10.4018/978-1-4666-2136-7.ch013
19. Benouini, R., et al.: Efficient image classification by using improved dual Hahn moment invariants. In: 2018 International Conference on Intelligent Systems and Computer Vision (ISCV) (2018)
20. Sayyouri, M., et al.: Improving the performance of image classification by Hahn moment invariants. J. Opt. Soc. Am. A Opt. Image Sci. Vis. **30**, 2381–2394 (2013)
21. Mukundan, R.: Some computational aspects of discrete orthonormal moments. IEEE Trans. Image Process. **13**(8), 1055–1059 (2004)
22. Spiliotis, IM, Mertzios, B.G.: Fast algorithms for basic processing and analysis operations on block represented binary images. Pattern Recognit. Lett. **17**, 1437–1450 (1996)
23. Spiliotis, I., Mertzios, B.: A fast parallel skeletonization algorithm on block represented binary images. Elektrik **1**, 161–173 (1997)
24. Spiliotis, I., Mertzios, B.: A fast skeleton algorithm on block represented binary images. In: 13th International Conference on Digital Signal Processing (DSP97), Santorini, Hellas (1997)
25. Gatos, B., Perantonis, S., Papamarkos, N.: Accelerated Hough transform using rectangular block decomposition. Electron. Lett. **32**, 730–732 (1996)
26. Spiliotis, I.M., Mertzios, B.G.: Real-time computation of two-dimensional moments on binary images using image block representation. IEEE Trans. Image Process. **7**, 1609–1615 (1998)
27. Spiliotis, I.M., Boutalis, Y.S.: Parameterized real-time moment computation on gray images using block techniques. J. Real-Time Image Process. **6**(2), 81–91 (2011)
28. Spiliotis, I.M., Boutalis, Y.: Fast and real-time moment computation methods of gray images using image block representation. In: Proceedings of 5th IASTED International Conference on Signal Processing, Pattern Recognition and Applications (SPPRA-2008), pp. 323–328, Innsbruck, Austria (2008)
29. Spiliotis, I.M., Karampasis, N.D., Boutalis, Y.S.: Fast computation of Hahn moments on gray images using block representation. J. Electron. Imaging (2020). https://doi.org/10.1117/1.JEI.29.1.013020
30. Chandra, R., Dagum, L., Kohr, D., Maydan, D., McDonald, J., Menon, R.: Parallel Programming in OpenMP. Academic Press, Cambridge (2001)
31. Barth, M., et al.: Best Practice Guide—Intel Xeon Phi (2014). http://www.prace-ri.eu/best-practice-guide-intel-xeon-phi-html/
32. Beyer, J., Larkin, J.: Targeting GPUs with OpenMP4.5 device directives. In: NVIDIA GPU Technology Conference, Silicon Valley (2016)
33. Szwoch, G., Ellwart, D., Czyzewski, A.: Parallel implementation of background subtraction algorithms for real-time video processing on a supercomputer platform. J. Real-Time Image Process. **11**, 111–125 (2016)

34. Hosny, K., et al.: Fast computation of 2D and 3D Legendre moments using multi-core CPUs and GPU parallel architectures. J. Real-Time Image Process. (2017). https://doi.org/10.1007/s11554-017-0708-1
35. Mahmoudi, R., Akil, M., Hedi, B.M.: Concurrent computation of topological watershed on shared memory parallel machines. Parallel Comput. **69**, 78–97 (2017)
36. Lu, Y., et al.: Parallelizing image feature extraction algorithms on multi-core platforms. J. Parallel Distrib. Comput. **92**, 1–14 (2016)
37. Spiliotis, I.M., Bekakos, M.P., Boutalis, Y.S.: Parallel implementation of the Image block representation using OpenMP. J. Parallel Distrib. Comput. **137**, 134–147 (2020). https://doi.org/10.1016/j.jpdc.2019.11.006
38. Camacho-Bello, C., et al.: Reconstruction of color biomedical images by means of quaternion generic Jacobi–Fourier moments in the framework of polar pixels. J. Med. Imaging **3**(1), 014004 (2016)
39. Hosny, K.M., Darwish, M.M.: Feature extraction of color images using quaternion moments. In: Recent Advances in Computer Vision: Theories and Applications. Springer (2019)
40. Quinn, M.J.: Parallel Programming in C with MPI and OpenMP. McGraw-Hill (2003)
41. Hutcheson, A., Natoli, V.: Memory Bound vs. Compute Bound: A Quantitative Study of Cache and Memory Bandwidth in High-Performance Applications. Stone Ridge Technology, Internal White Paper (2011)
42. Gentile, A., Sander, S., Wills, L., Wills, S.: The impact of grain size on the efficiency of embedded SIMD image processing architectures. J. Parallel Distrib. Comput. **64**, 1318–1327 (2004)
43. Intel Corporation: Avoiding and Identifying False Sharing Among Threads (2011). https://software.intel.com/en-us/articles/avoiding-and-identifying-false-sharing-among-threads
44. Karp, A.H., Flatt, H.P.: Measuring parallel processor performance. Commun. ACM **33**, 539–543 (1990)
45. Agustsson, E., Timofte, R.: NTIRE 2017 challenge on single image super-resolution: dataset and study. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017)
46. DIV2K Dataset: DIVerse 2K Resolution High Quality Images as Used for the Challenges @ NTIRE (CVPR 2017 and CVPR 2018) and @ PIRM (ECCV 2018). https://data.vision.ee.ethz.ch/cvl/DIV2K/
47. Otsu, N.: A threshold selection method from gray-level histograms. IEEE Trans. Syst. Man Cybern. **9**(1), 62–66 (1979)
48. Chen, S., Dongarra, J., Hsiung, C.: Multiprocessing linear algebra algorithms on the CRAY X-MP-2: experiences with small granularity. J. Parallel Distrib. Comput. **1**, 22–31 (1984)