GLCM Based Parallel Texture Segmentation using A Multicore Processor

Shefa Dawwd
Department of Computer Engineering, Mosul University, Iraq

Abstract: *This paper investigates the using of Gray Level Co-Occurrence Matrix (GLCM) based on supervised texture segmentation. In most texture segmentation methods, the processing algorithm is applied to a window of the original image rather than to the entire image using sliding scheme. To attain a good segmentation accuracy especially in the boundaries, optimal size of window is determined, or windows of variant sizes are used. Both options are very time consuming. Here, a new technique is proposed to build an efficient GLCM based texture segmentation system. This scheme uses a fixed window of variant apertures. This will reduce the computation overhead and recourses that required to compute GLCM, and will improve the segmentation accuracy. Image's windows are multiplied with a matrix of local operators. After that, GLCM is computed and features are extracted and classified and the segmented image is produced. In order to reduce the segmentation time, two similarity metrics are used to classify the texture pixels. Euclidean metric is used to find the distance between the current and previous GLCM. If it is above a predefined threshold, then the computation of GLCM descriptors are required. Gaussian metric is used as a distance measure between two GLCM descriptors. Furthermore, a median filter is applied to the segmented image. Finally, the transition and misclassified regions are refined. The proposed system is parallelized and implemented on a multicore processor.*

Keywords: *GLCM, haralick descriptors, median filter, moving window, texture segmentation.*

Received November 13, 2014; accepted September 19, 2016

1. Introduction

Texture segmentation is one of the most image segmentation challenging problems. It is difficult to choose a suitable mathematical model to describe variant types and sizes of primitive textures. From the computer vision point of view, it is difficult to recognize significant edges within the non-homogeneous intensity distributions of texture image. Another issue is that the primitive texture is usually unstable against changes in scale, translation and rotation.

To cope with after challenges, it is crucial to build an efficient texture descriptor (feature extractor) that is well describes texture. In addition, an efficient classification criteria should be followed to segment the descriptors to different texture classes.

Typically, image is evaluated by using one of the two approaches. In the first approach a different pyramidal scales of the original image is utilized [12, 14, 15] to determine in which scale of the pyramid the texture primitive is included.

Image segmentation pyramids is classified into regular and irregular types. Although that the pyramidal approach based segmentation is invariant against noise, the regular type suffers from many limitations: the most important is incapability to segment elongated objects. To address the limitations, irregular type is proposed. However, the irregular type is more complex and time consumed to be achieved [1]. The second approach evaluates the original image over a specified Region Of Interest (ROI) or window [3, 5, 11, 18]. This approach is more suitable for processing image that is composited of elongated regions. Window size has a significant effect on the segmentation accuracy. Large window size leads to more stable texture features but tends to blur the edges, while small window size leads to misclassify the textured boundaries. Thus, an efficient windowing should be achieved. A good window is the one that leads to discriminate among variant texture primitives and to achieve a sufficient texture representation within it.

In [3], the optimal window size for Gray Level Co-Occurrence Matrix (GLCM) descriptors according to the texture that is to be classified is determined by using semivariogram method. 13×13 and 7×7 pixels windows are determined to be optimal for the building and vegetation sample images respectively. Ouma *et al*. [9] investigate the use of an optimal window size for wavelet based textured feature extractor to classify urban tree texture.

The experiments presented in [10] shows that the integration of multi-sized windows yields lower classification errors than when optimal single-sized windows are considered. A coarse-to-fine strategy is used in [17] to generate an ideally step-like transition closer to a dashed vertical line by adjusting the window size adaptively from 27×27 to 9×9. The adjustment is applied only in the target area of transition boundary. While 3×3, 5×5, and 7×7 windows are applied to TerraSAR-X (TSX) images in the work presented in [7], in order to define the most suitable parameters discriminating between different land cover types.

The original 160×120 image is divided into multiple windows of 80×60, 40×30, and 20×15 in the work presented in [8]. The GLCM of different parameters is applied to each of these windows and a set of features are extracted together from these windows and the textures are classified by their features using machine learning approaches.

In the works presented above, it can be noticed that different window sizes and image resolutions, are employed. The whole segmentation algorithm is applied to each window or image resolution. This add more computational power and time overhead. Time reduction becomes a crucial requirement especially for fast processing of video sequence or large medical image. Therefore, the goal of this paper is to achieve a good segmentation in reduced computation time and variant parameters. The paper proposes new techniques to solve the aforementioned issues:

- Using of local operators: instead of using variant sizes of windows, fixed size of window with variant aperture can be used. Consequently, the same algorithm can be used to process texture of variant primitives and a extra modification of the algorithm parameters is avoided and then more stable features are extracted.
- To reduce the segmentation time, two metrics are used to classify the texture pixel. The first one (Euclidean) check if the adjacent GLCMs are close enough, then redundant computation of GLCM descriptors are avoided. Otherwise, Gaussian metric is used to find distance between two vectors of GLCM descriptors.
- Further refinement is achieved by using median filtering of the image that is already segmented.
- Further reduction of segmentation time is achieved by parallelizing the proposed algorithm among multiple cores of the host processor.

2. Texture Description Using GLCM

The GLCM texture analysis method is investigated in this paper. Haralick *et al.* [4], defined the GLCM as a one of the best known texture analysis methods. The selection of GLCM is based on some attractive features that extracted from previous studies. GLCM has been used in many applications, such as in content based image retrieval, biomedical, etc. Furthermore, GLCMs of an original image is approximately the same with GLCMs of its rotation. That is based on using four directions for each descriptor. The most common statistic in medical field is co-occurrence texture models, which demonstrates better classification accuracy [16]. The co-occurrence matrix includes second-order grey-level information, which is mostly related to human perception and the discrimination of textures. As presented in [13], it performs the best among all texture models.

GLCM is a statistical texture analysis method which deals with supervised texture segmentation in a frame partition using level-set deformable model implementation.

The GLCM is computed in a user defined moving window rather than computing it for the entire image. Using a moving window, neighborhoods of the pixel are defined and texture features for each window in an image is computed. The GLCM and its computation along moving window can be represented in mathematical notation as:

$$P_{ij}(a) = p_{ij}(I(r+a)) \qquad r \in D \tag{1}$$

where:
I : is the input image.
r: is a 2D position within the moving window.
D: is the moving window within the image.
a: is the displacement moving step.
p_{ij}: is the GLCM which defined as:

$$\sum_{x=1}^{n_D}\sum_{y=1}^{n_D}\begin{cases}1, & if \qquad I(x,y)=i \,\&\, I(x+\Delta_x, y+\Delta_y)=j\\ 0, & otherwise\end{cases}$$

[Δ_x, Δ_y]: is specifying the offset between the pixel-of-interest located at (x,y) and its neighbor. It takes four options: [0, d], [-d, d], [-d, 0], [-d, -d].
n_D: is the dimention of the moving window $i,j \in \{0,1,2,\ldots\ldots,L-1\}$, L: level of gray tone.

The number of possible intensity levels in the image determines the size of GLCM. For an 8-bit image (256 possible levels), GLCM will be of size 256×256. This is not a problem when working with one matrix, but co-occurrence usually used in sequences. In order to reduce computation load, an approach used frequently is to quantize the intensities to limited levels to keep the size of GLCM manageable. For example, in the case of 256 intensities we can do this by letting the first 32 intensity levels equal to 1, the next 32 equal to 2, and so on. This will result in a co-occurrence matrix of size 8×8.

After the formulation of GLCM, a set of descriptors useful for characterizing the content of GLCM is to be computed. Some of these descriptors that are used in this paper is defined as follows [4]:

Correlation: $$\sum_{i=1}^{k}\sum_{j=1}^{k}\frac{(i-m_x)(j-m_y)P_{ij}}{\sigma_x\sigma_y} \tag{2}$$

Contrast: $$\sum_{i=1}^{k}\sum_{j=1}^{k}(i-j)^2 P_{ij} \tag{3}$$

Homogeneity: $$\sum_{i=1}^{k}\sum_{j=1}^{k}\frac{P_{ij}}{1+|i-j|} \tag{4}$$

Entropy: $$-\sum_{i=1}^{k}\sum_{j=1}^{k}P_{ij}\,\log P_{ij} \tag{5}$$

Energy: $$\sqrt{\sum_{i=1}^{k}\sum_{j=1}^{k} P_{ij}^{\;2}} \qquad (6)$$

Where m_x, m_y and σ_x, σ_y denote the mean and standard deviations of the row and column sums of the matrix P_{ij}, respectively.

In order to capture all possible texture patterns in a further step, different displacements (d=1, 2, 3, 4,….) for four directions (θ =0°, 45°, 90°, 135°) are evaluated. In this paper, five texture descriptors (Equations (2, 3, 4, 5, 6)) are computed for each directions. Therefore, for each texture displacement, 20 descriptors (5 descriptors×4 direction) are computed. It is found that the more the number of descriptors, the more complicated textured primitive is specified, but at the same time the generalization is reduced, particularly in the transition regions. Therefore, a moderate number of descriptors is selected.

3. The Proposed Technique

The main steps for image segmentation system is presented in the following algorithm:

Algorithm 1: GLCM based texture segmentation algorithm

```
1. Subdivide the texture image into n-overlapping
slides, select the aperture size, pattern and slide
moving steps.
2. Reset GLCM_i, Initialize Gray_i, moving step i=0.
3. Start with new slide index: i.
4. Compute GLCM_i.
5. If dist(GLCM_i, GLCM_{i-1})<threshold, then assign
 each of the slide pixel to Gray_{i-1} and go to 9 else
 continue.
6. Feature extraction: calculate GLCM descriptors
vector (≤N descriptors).
7. Calculate distance between slide vector with a bank
of reference feature vectors (each of N descriptors).
8. Classification: assign each of the slide pixel to one
gray level (label of the closest reference vector).
9. If slide index < n, then go to 3 else continue.
10. Apply traditional median filter.
11. If the segmentation quality is poor, then change the
 aperture size and go to 2, else continue.
12. End.
```

Here, the moving window is scanned over the image in overlapped steps. To avoid using windows of different sizes, a fixed window of variant aperture is used. This can be achieved by multiplying the input window with a matrix of local operators (W) of ones and zeros. Now, Equation (1) can be rewritten as:

$$P_{ij}(a) = p_{ij}(I(r+a)\cdot W) \qquad (7)$$

The distribution of ones and zeros in W can define the pattern of the window and its aperture. In Figure 1, one can see that the moving window is not restricted of its traditional first order pattern of neighbourhood (Figure 1-a). But it can take any other pattern (Figure 1-d). The one's operators define the size of the aperture (Figure 1-b, and 1-c).

$$\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & \mathbf{1} & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{bmatrix}$$

a) First order neighbourhood.

$$\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & \mathbf{1} & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

b) Second order neighbourhood.

$$\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & \mathbf{1} & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

c) Third order neighbourhood.

$$\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & \mathbf{1} & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

d) Fourth order neighbourhood.

Figure 1. Neighbourhood pattern with central pixel.

Since the operators are only ones or zeros, no multiplication operation is required for weighting the input (I) in the region D (in hardware AND operation can be used).For each window in local position a, GLCM is computed. Then Features of each window are extracted (the GLCM N-descriptors). To reduce the computational overhead, different approaches are used:

1. The intensity level of the input texture image is quantized to 20 levels. Then instead of 256×256 elements of GLCM, 20×20 elements GLCM is produced.
2. When computing texture descriptors by scanning the window slide throughout an image, shifting the window one step in an overlapping fashion does not change many of the GLCM elements, particularly inside the same texture class. Therefore, if the distance between the current and the previous GLCM is too low, the distance between their descriptors is also expected to be too low. Then there is no need to re-compute the GLCM descriptors vector of the current window and the previous gray level can be assigned to it.
3. The distance between the extracted feature with the reference features which obtained in a supervised fashion is computed. Each reference vector that is stored in the feature vectors bank should have an adequate number of elements (descriptors). The moving window may be described with a smaller number of descriptor (< N: select good descriptors from each vector) to decrease the time of computations.

The distance measure is crucial factor for good segmentation quality for both inter and intra class regions. For more discussion about the similarity metrics, dealing with the transition regions and refinements, the following subsections are presented.

3.1. Similarity Measure

In this paper two distance metrics are used: the Euclidean distance and Gaussian curve. To find the distance between two GLCMs, and since that all elements of the GLCM are homogeneous (totally integer values, or totally real values in a normalized GLCM), the Euclidean metric can be used. On the other hand, we find that the most popular Euclidean or Manhattan metrics may not be suitable to find the closest reference vector with the descriptors vector of the slide under the test. This is due to the distribution heterogeneity of each of the GLCM descriptor. Thus, a Gaussian curve is employed to find similarity measure between two vectors. The mathematical representation is defined as:

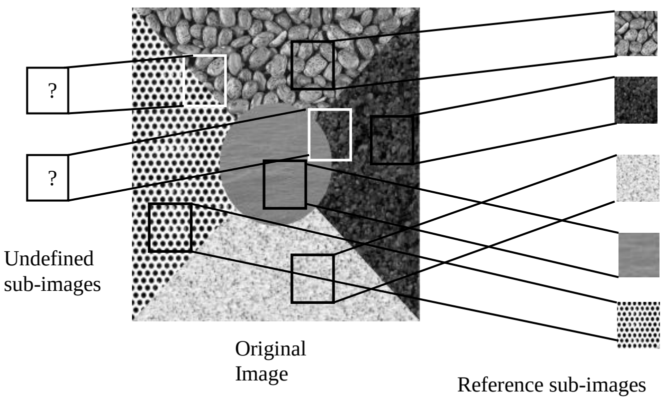
$$f\,(\varphi) = e^{\frac{-(c-\varphi)^2}{2\sigma^2}} \qquad (8)$$

Where ϕ is a descriptor value. The c and σ are the centre and width of Gaussian curve, respectively. For each GLCM descriptor, Note that the similarity ranges is from 0 to 1. It equals to 0 for two completely dissimilar descriptors, and 1 for descriptors with matching central. An input vector is mapped to all the reference vectors. Each feature of this vector is mapped to all specified features of reference vectors. Then the sum of all Gaussian outputs among reference vectors and input one are calculated. Finally, the input vector is assigned to the reference vector where the summation of Gaussian outputs is the maximum. The former metric is considered good if the values it assigns to similar pairs of GLCM are consistently lower than the values it assigns to dissimilar GLCM. In contrast, the later metric is considered good if the values it assigns to similar pairs of GLCM descriptors are consistently higher than the values it assigns to dissimilar GLCM descriptors. For both metrics, a threshold (or an interval) is assigned beforehand, such that metric values above the threshold (interval) indicate similarity and below the threshold (interval) indicate dissimilarity.

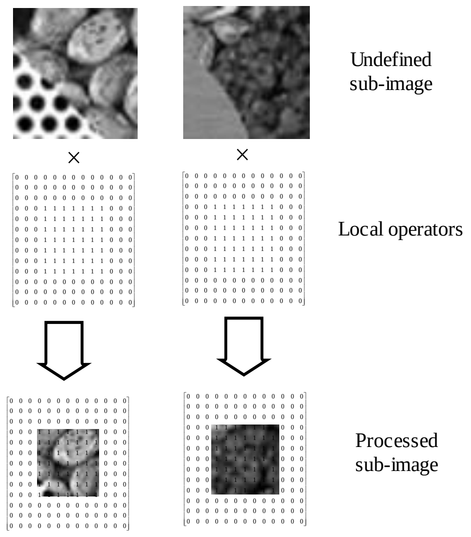
3.2. Transition Region and the Refinement

The transition between two or more completely different classes of texture is known as a transition region. Two techniques deal with transition region. The first technique depends on using new shapes of the moving window in the feature extraction stage. To explain our approach let's begin with the standard texture image downloaded from Brodatz dataset [1]. The images are digitized at a resolution of 256×256 pixels and at 8-bit grey scale level. This image contains five different textures joined with sharp transition regions. First, according to the algorithm presented in Algorithm 1, clear reference sub-images are extracted from the original image (Figure 2-a).

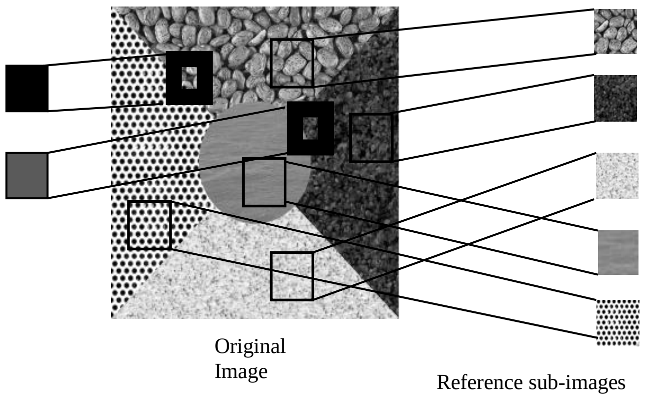
In Figure 2-a, one can see that a rectangular pattern is used to extract each reference or extracted sub-image. Each of sub-image depends on the weights of the window. If instance, a 13×13 slice is used, then the weights shown in Figure 1-a represents the extraction without any mask. Now, if the first sub-image from the original image (which contains two textures) is extracted, its feature vector cannot be clustered in any of the five referenced feature vectors correctly, because it is assumed to be as a new class. The size of the window should be small enough to reduce this effect.



a) Broadatz texture image and its extracted and reference sub-images.



b) Processing of undefined sub-images using local operators.



c) Segmentation of the processed sub-images.

Figure 2. Segmentation process.

But this may lead to increase the misclassification rate in regions other than the transition edges. We use local operators to avoid using windows of different sizes. These operators are adapted to function the behavior of window with a variant aperture.

Now, if the window's aperture becomes smaller enough (Figure 1-b and 1-c), keeping in mind that the window size is constant, then, the Haralick descriptors can match the descriptors of the first reference sub-image, and then the pixel can exactly classified to the first class. Processing and segmentation of undefined windows with local operators are shown in Figure 2-b and Figure 2-c respectively.

The window aperture can take different shapes with different aperture sizes as shown in Figure 1-d and 1-e. The influence of using such shape is to sharpening the transition regions of rotated angles (45^o, 135^o). Note that there is no time overhead imposed by using the above technique.

The second technique is illustrated in algorithm 1, where a traditional median filtering is applied to the already segmented image. Afterword, the transition regions and misclassified spots within the truly segmented region are refined. The above algorithm well works with window of smaller size.

4. Parallel Implementation of The Proposed Algorithm

Over the past decade, computing architectures have started on a clear trend towards increased parallelism and heterogeneity, with most mainstream microprocessors now including General Purpose Multi-Cores (GPMCs), and system architectures commonly integrating accelerators such as Graphics-Processing Units (GPUs) and Field Programmable Gate Arrays (FPGAs) over PCI and even on the same chip. Numerous studies have shown that such architectures can accelerate applications by orders of magnitude compared to sequential software [2].

Although, using a specific hardware on FPGA is expected to give best performance in term of speed and real time applications, and using FPGA is the most flexible hardware choice according to the feature of re-programmability, however, the software based implementations are more flexible and their development time are much less than the FPGA. Therefore, implementing the mentioned algorithms on a multi core parallel architecture such as GPMC or GPU is an excellent alternative to provide moderate performance**.** It is important to emphasise that GPU is not accessible for all researchers. Therefore, GPMC can be selected as an initial platform toward more powerful implementation (GPU or FPGA).

Starting from 2004, GPMCs are replacing traditional Central Processing Units (CPUs) in both personal computers and servers. Generically called “multi-cores”, they are already offered by most of the big players-Intel, Sun, Advanced Micro Devices (AMD), and IBM. GPMCs are homogeneous platforms with complex cores, based on traditional processor architectures; they are typically shared-memory architectures, with multiple layers of caches, and they are used as stand-alone processors [6]. The purpose of using multi cache levels is to reduce cores access to the global shared memory as much as possible.

The main drawback of GLCM is that the computation of the GLCMs and texture features are computationally intensive and time-consuming. In this paper we focus our implementation on using GPMCs as a type of multi-cores programmable parallel machines.

MATLAB is one of the most commonly popular languages used in technical computing. It is prefer to develop an algorithm in Matlab first. Then, it can be converted the code into C or FORmula TRA Nslation (FORTRAN) for real life demands. The MATLAB Distributed Computing Server (MDCS) offered by the Math WorksInc along with Parallel Computing Toolbox (PCT) provides tools and routines for parallelizing sequential tasks. The PCT provides functionality to run MATLAB code on multicore systems and clusters. Such as parallel for-loop execution, or creation/manipulation of distributed arrays as well as message passing functions for implementing fine grained parallel algorithms. Parallel Matlab environment is used to write a parallel program for GLCM computation using the mentioned multi-core processor.

A parallel MATLAB program is written by using either parfor loops statement or spmd. The parfor approach is a limited but it is a simple way to distribute the work over the available cores where the MDCS automatically distribute the load balance among the available cores. The spmd statement is powerful, but requires rethinking the program and data, then the user can manually distribute the load balance for a each specific core (see Figure 3).

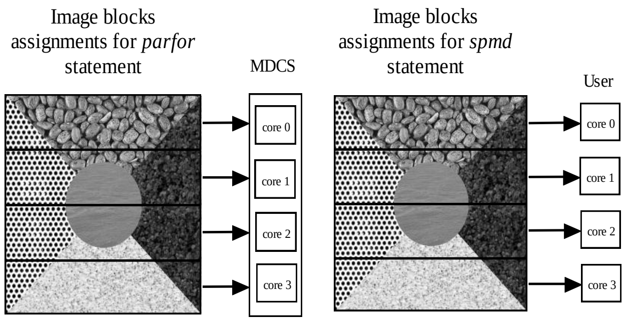


Image blocks assignments for *parfor* statement
MDCS
core 0
core 1
core 2
core 3
Image blocks assignments for *spmd* statement
User
core 0
core 1
core 2
core 3

Figure 3. Parallel distribution of image partitions if four cores are available.

Also, a standard parallel programming library based on the fork-join parallel paradigm: OpenMP is used. The C language with the fork-join model of OpenMP is a suitable choice for multi-threading the data shown in Figure 3.

5. Results and Discussions

The proposed system is implemented on General Purpose Multi-Cores (GPMCs) platform. A laptop provided with a processor type of Intel® Core™ i5 2.4GHz is used for segmentation using GLCM.. This parallel platform is programmed by using either MATLAB parallel paradigm or C-language supported by OpenMP shared memory parallel paradigm.

The segmentation result of the texture image shown in Figure 2 is presented in Figures 4 and 5. In every part of Figure 4, the segmentation results before and after using of median filter are shown. The misclassified pixels and the percentage of error (E%) over the best segmented image achieved in this paper are also presented. It can be noticed that in all parts of the Figure 6, using median filter reduces the misclassified pixel and E%. In Figure 4-a (top), a degraded result and large misclassified pixel and segmentation error is clearly shown when smaller window size is used (local operators are all 1's). The same can be seen in Figure 4-a (middle) with window of medium size, but some improvements inside the texture classes are achieved. Although that larger size of window will improve the inter classes segmentation quality, but this leads to degrade the intra classes quality (the transition regions or borders).

In Figure 4-a (bottom), our proposed algorithm is used but with the Euclidean similarity metric for descriptors assignment. A segmentation deformation can be noticed even with the filtered version. The effectiveness of the proposed algorithm when the Gaussian metric is used is shown in Figure 4-b. Note that a big improvement of the segmentation quality is achieved. The best results is achieved with aperture of size of 9×9 pixels (Figure 4-b (middle)) in comparison with either too big (Figure 4-b (top)) and too small (Figure 4-b (bottom)) aperture sizes. The output image in this case is chosen as a reference to test the other cases, since that it is hard to achieve a segmented image that is totally free of misclassified pixels. Thus, E% is approximated to zero. As it can be noticed, a relatively small size of window and aperture is used to reduce the transition region effect. At the same time using median filter refines the image and reduce the effect of choosing these small sizes inside the texture classes.

The results when using both similarity metrics are shown in Figure 5-a. As we mentioned earlier, the goal is to reduce the computation time and to preserve the segmentation quality as possible. Segmented images with low misclassified pixel and low E% are achieved when different threshold values are employed. Better transition region refinement are achieved in comparison to some works [11, 19] (Figure 5-b) that used the same testing image. Depending on the application, the threshold can be chosen as a trade-off between the time and the required quality. The segmentation time is reduced to about half when the error reaches to 4.86 (Figure 6-a).

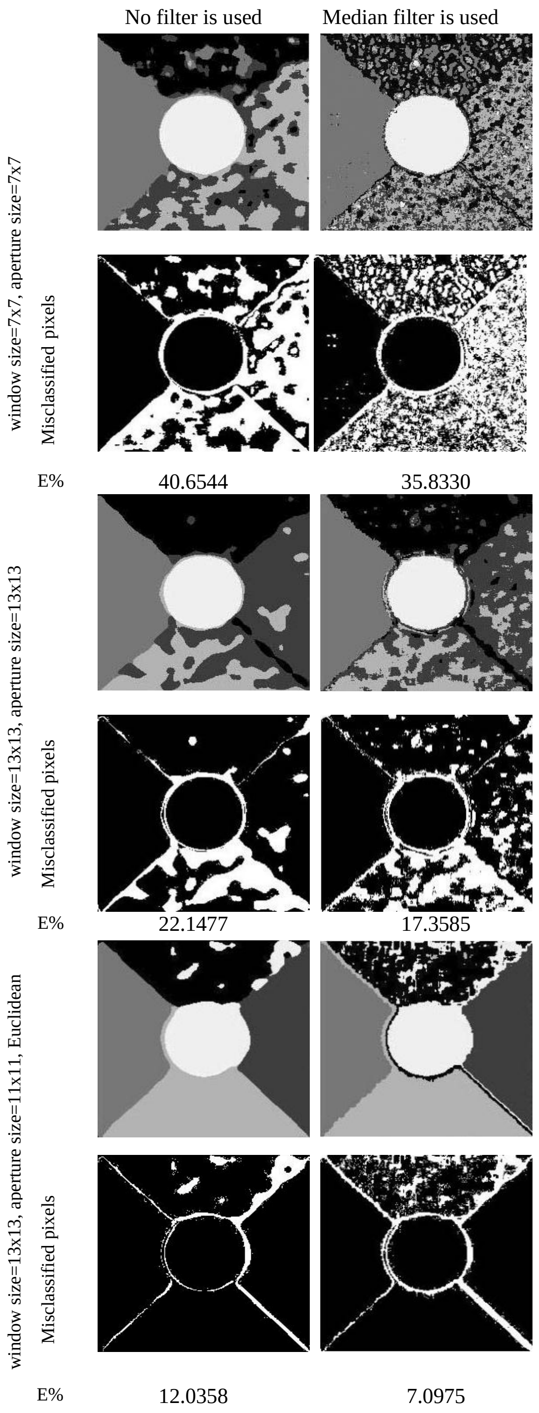
The segmentation time based on calculating GLCM with 256×256 Broadatz texture image in Matlab and OpenMP C-program is shown in Figure 6-b. By examining the chart of six cores, it can be noticed that the best performance is achieved when three or four cores are employed. When employing five or six cores, the execution time cannot be reduced but even increased due to the memory access overhead issue. This is why the chart goes up after more cores are employed. A longer segmentation time is consumed when using parfor statement in comparison to the time consumed when using spmd. The core assignment overhead is released when explicit assignment is performed in spmd rather than the automatic assignment that performed when using perfor statement.

The proposed method gives almost a 3× performance gain as a speedup. The overall speedup can be computed from the time results as a ratio between sequential execution time and parallel execution time. The speedup is achieved due to the reduction of computations by decreasing the required descriptors, and by distributing computations over the multiple processor cores. Based on the results of the technique of time reduction proposed in this paper and presented in Figure 6-a, the above speedup can be duplicated. A considerable reduction of segmentation time is achieved in comparison with works mentioned in the literature.

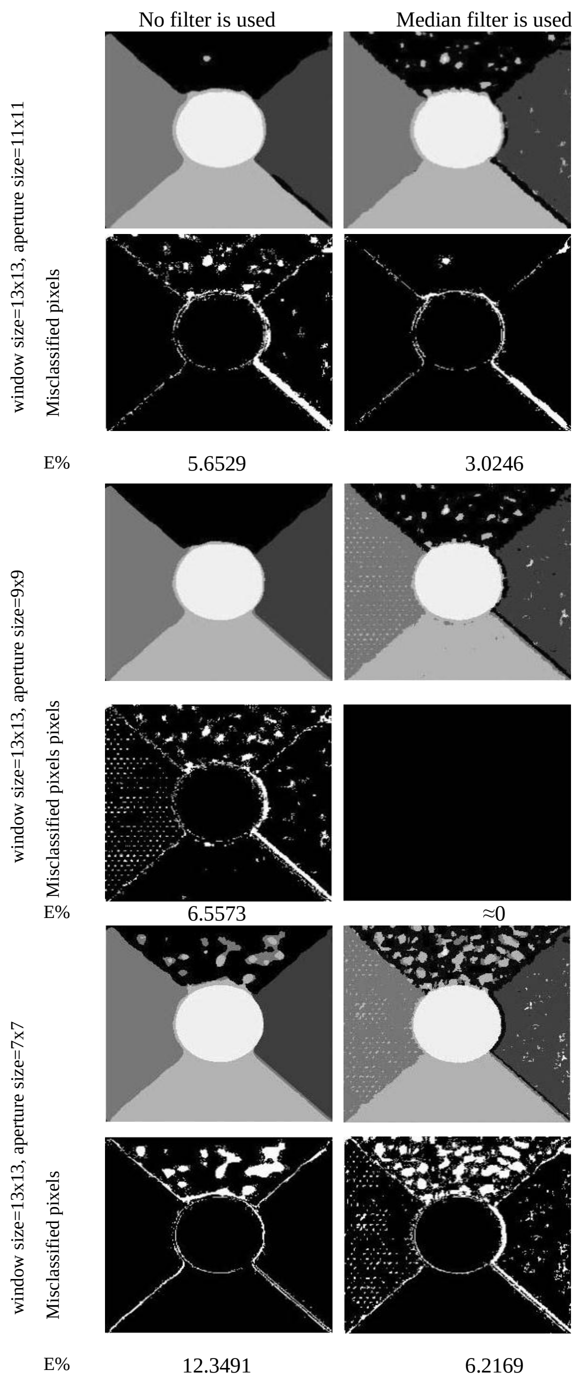
6. Conclusions and Future Works

New techniques are employed to build an efficient GLCM based texture segmentation system using a fixed window of variant apertures. A matrix of local operators are multiplied with the window slide. Then the most accurate class of the texture region that is to be classified is determined especially in transition regions. A well segmentation quality is achieved either within-texture class variance or between-texture class variance. A reasonable reduction in segmentation time is accomplished when Euclidean distance metric is used to determine whether the current GLCM is similar to the previous one. If it is attained, then redundant calculations of GLCM descriptors are avoided. Using Gaussian similarity metric to find distance between two vectors of GLCM descriptors, gives a good classification accuracy. A considerable refinement of the segmentation quality is achieved when median filter is applied to the image that is already segmented. The resulted speedup can be duplicated according to a threshold value that should be selected depending on the application. A relatively small size of sliding window can be used with the above technique and gives acceptable results. Also, extra modification of algorithm parameters are avoided due to use of fixed size of moving window.

These two features are very desirable if the algorithm proposed is implemented in hardware for higher performance as it may be used with video image. The latter is left for future work.

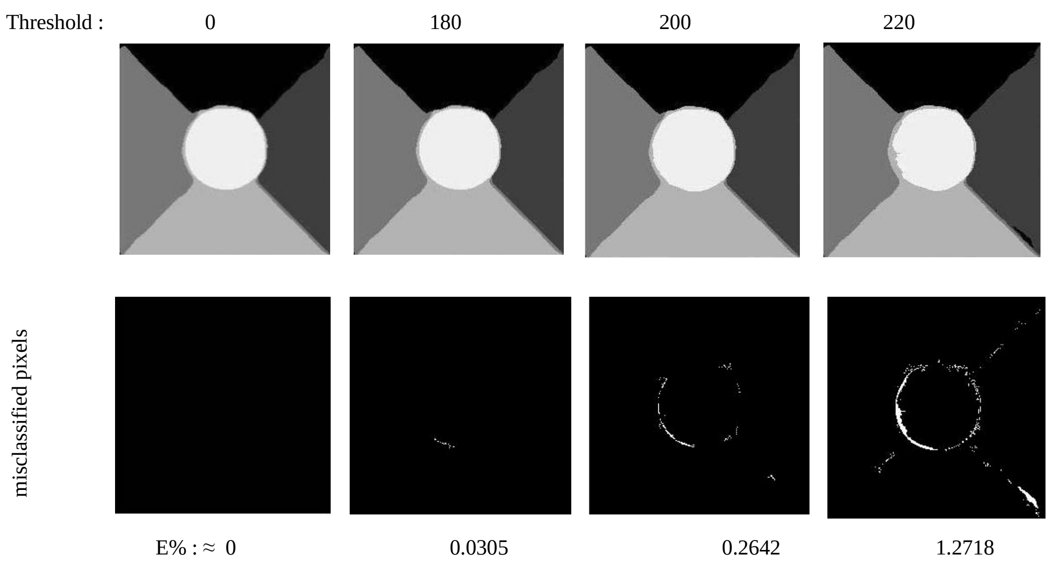


a) The proposed algorithm is partially applied.

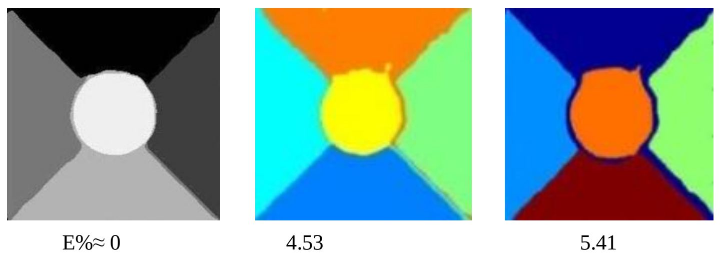

No filter is used
Median filter is used
window size=13x13, aperture size=11x11
Misclassified pixels
E%
5.6529
3.0246
window size=13x13, aperture size=9x9
Misclassified pixels pixels
E%
6.5573
≈0
window size=13x13, aperture size=7x7
Misclassified pixels
E%
12.3491
6.2169

b) The proposed algorithm is wholly applied using Gaussian similarity metric.

Figure 4. Segmentation results with variant parameters and classification criteria's.

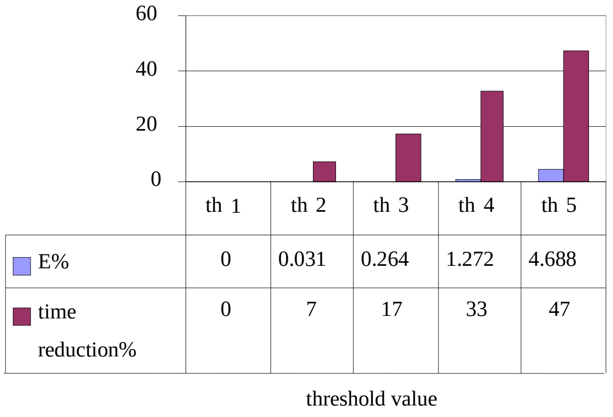


a) Segmentation results when the proposed algorithm is wholly applied using both similarity metrics (window size=13×13, aperture size=9×9) for different threshold values with median filtering**.**



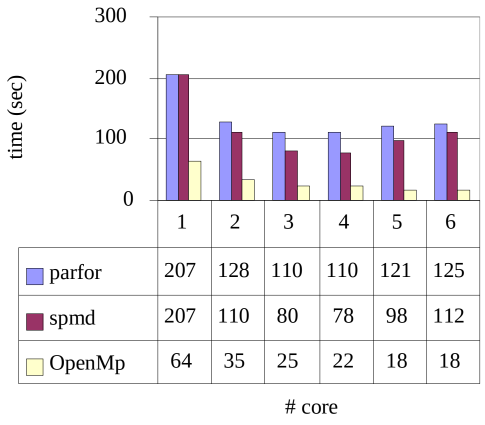
b) Comparison of the proposed algorithm (left) with [11] (middle) and [19] (right).

Figure 5. Segmentation results with comparison to previous works.



	th 1	th 2	th 3	th 4	th 5
🟦 E%	0	0.031	0.264	1.272	4.688
🟥 time reduction%	0	7	17	33	47

a) Classification error and time reduction variation with different threshold values.



	1	2	3	4	5	6
🟦 parfor	207	128	110	110	121	125
🟥 spmd	207	110	80	78	98	112
🟨 OpenMp	64	35	25	22	18	18

b) Parallelizing GLCM segmentation of 256×256 Broadatz texture image in Matlab and C-program using *parfor* statement, *spmd* statement and C-language with OpenMP.

Figure 6. The segmentation processing time.

References

[1] Brodatz P., *A photographic Album for Artists and Designers*, Dover New York, 1996.

[2] Fowers J., Brown G., Cooke P., and Stitt G., "A Performance and Energy Comparison Of Fpgas, Gpus, and Multicores for Sliding-Window Applications," *in Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, Monterey, pp. 47-56, 2012.

[3] Giannini M., Merola P., and Allegrini A., "Texture Analysis for Urban Areas Classification in High Resolution Satellite Imagery," *Applied Remote Sensing Journal*, vol. 2, no. 2, pp. 65-71, 2012.

[4] Haralick M., Shanmugam K., and Dinstein I., "Textural Features for Image Classification," *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-vol. 3, no. 6, pp. 610-621, 1973.

[5] Huang X., Zhang L., and Li P., "A Multiscale Feature Fusion Approach for Classification of Very High Resolution Satellite Imagery Based on Wavelet Transform," *International Journal of Remote Sensing*, vol. 29, no. 20, pp. 5923-5941, 2008.

[6] Kanter D., "Inside Nehalem: Intel´s future processor and system,"

http://www.realworldtech.com/ page.cfm? ArticleID=RWT040208182719, Last Visited, 2008.

[7] Mahmoud A., Elbialy S., Pradhan B., and Buchroithner M., "Field-Based Land Cover Classification Using Terra SAR-X Texture Analysis," *Advances in Space Research*, vol. 48, no. 5, pp. 799-805, 2011.

[8] Mohamed A. and Lu J., "Analysis of GLCM Parameters for Textures Classification on UMD Database Images," *in Proceedings of The 5th International Conference on Advanced Communications and Computation*, Brussels, pp. 111-116, 2015.

[9] Ouma Y., Ngigi T., and Tateishi R., "On the Optimization and Selection of Wavelet Texture for Feature Extraction from High-Resolution Satellite Imagery with Application Towards Urban-Tree Delineation," *International Journal of Remote Sensing*, vol. 27, no. 1, pp. 73-104, 2006.

[10] Puig D. and García M., "Pixel-Based Texture Classification by Integration of Multiple Texture Feature Evaluation Windows," *in Proceedings of Iberian Conference on Pattern Recognition and Image Analysis*, Puerto de Andratx, pp. 793-801, 2003.

[11] Rampun A., Strange H., and Zwiggelaar R., "Texture Segmentation Using Different Orientations of GLCM Features," *in Proceedings of 6th International Conference on Computer Vision/Computer Graphics Collaboration Techniques and Applications*, Berlin, 2013.

[12] Schroeter P. and Bigun J., "Hierarchical Image Segmentation by Multi-Dimensional Clustering and Orientation Adaptive Boundary Refinement," *Pattern Recognition*, vol. 28, no. 5, pp. 695-709, 1995.

[13] Sharma M. and Singh S., "Evaluation of Texture Methods for Image Analysis," *in Proceedings of the 7th Australian and New Zealand Intelligent Information Systems Conference*, Perth, pp. 117-121, 2001.

[14] Siqueira F., Schwartz W., and Pedrini H., "Multi-Scale Level Co-Occurrence Matrices for Texture Description," *Neurocomputing*, vol. 120, pp. 336-345, 2013.

[15] Stojmenovic M., Montero A., and Nayak A., "Colour and Texture Based Pyramidal Image," *International Conference on Audio Language and Image Processing*, Shanghai, pp. 778-786, 2010.

[16] Susomboon R., Raicu D., Furst J., and Johnson T., "A Co-Occurrence Texture Semi-Invariance to Direction, Distance and Patient Size," *in Proceedings of SPIE-The International Society for Optical Engineering*, San Diego, 2008.

[17] Tsai Y., "Texture Image Segmentation Using Adaptive Gray Level Co-Occurrence Probabilities," *in Proceedings of the International Conference on Image Processing, Computer Vision, and Pattern Recognition*, Las Vegas Nevada, 2008.

[18] Wang H., Feng Y., Sa Y., Lu J., Ding J., Zhang J., and Hu X., "Pattern Recognition and Classification of Two Cancer Cell Lines by Diffraction Imaging at Multiple Pixel Distances," *Pattern Recognition*, vol. 61, pp. 234-244, 2017.

[19] Zheng Y. and Chen K., "A Hierarchical Algorithm for Multiphase Texture Image Segmentation," *ISRN Signal Processing*, vol. 2012, pp. 1-11, 2012.

Shefa Dawwd was born in Mosul-Iraq in 1970. He received the B.Sc degree in electronic and communication Engineering, the M.Sc and the Ph.D degree in computer Engineering in 1991, 2000, and 2006, respectively. He is presently a faculty member (Associate Professor) in the computer engineering department/University of Mosul. His main research interests include image & signal processing and their hardware models, parallel computer architecture, hardware implementation and GPU based systems. He has authored more than 30 research papers and book chapters. He has been an editorial member of several national and international journals.