MULTIRESOLUTION SEGMENTATION: A PARALLEL APPROACH FOR HIGH RESOLUTION IMAGE SEGMENTATION IN MULTICORE ARCHITECTURES

P. N. Happ [a], R. S. Ferreira [a], C. Bentes [b], G. A. O. P. Costa [a], R. Q. Feitosa [a]

[a] Department of Electrical Engineering, Pontifical Catholic University of Rio de Janeiro (PUC-Rio)
Rua Marquês de São Vicente 225, Gávea, CEP 22453-900, Rio de Janeiro, RJ, Brazil
(patrick, rsilva, gilson, raul)@ele.puc-rio.br
[b] Dept. of Computer and Systems, Rio de Janeiro State University (UERJ)
Rua São Francisco Xavier 524, Maracanã, CEP 20550-900, Rio de Janeiro, RJ, Brazil
cristianabentes@gmail.com

KEY WORDS: Remote Sensing, Image Processing, Parallel Processing

ABSTRACT:

In automatic image interpretation, the process of extracting different objects that compose an image is one of the primary steps. This process is known as image segmentation and consists of subdividing an image into meaningful regions, also called segments, which will be classified in a later step. Many of the existing segmentation algorithms, however, have high computational cost for large images as the currently high-resolution remote sensing images. The main focus of this paper is to tackle this problem by using parallel processing. The idea is to explore current multi-core architectures available in commercial processors in order to speedup the segmentation process. A multithreading parallel implementation of a region growing algorithm proposed originally by Baatz and Schäpe (2000) is presented that aims at providing better execution times, while delivering a similar outcome produced by the sequential version. The algorithm is able to work with any number of threads, which is defined as an input parameter, so as to take full advantage of the upcoming processors having any number of cores. The current parallel implementation was tested on three different images on a quad-core processor and obtained up to 2.6 of segmentation speedup.

1. INTRODUCTION

The image segmentation procedure has been an issue widely discussed in the field of digital image processing and computer vision. Segmentation algorithms for region growing group pixels or sub regions into larger regions, based on a set of initial points (called seeds) that grow annexing adjacent regions that have similar properties (e.g. texture or color). This class of algorithms has been usually applied especially in the remote sensing area. The disadvantage is the high computational cost for large images (Wassenberg et. al., 2009).

The aim of this paper is to develop a parallel implementation for the image segmentation algorithm proposed in (Baatz et al., 2000). The idea is to harness the parallel processing capability present in most modern processors, specifically the multiple computing cores in one processor. Therefore, the proposed solution does not require special hardware and can run on low-cost machines that are commercially available.

The parallel implementation is based on the division of the process into threads. Since the segmentation quality is a crucial step for classification (Blaschke et al., 2001, Pal et al., 1993), the parallelization process should not compromise the results. Another concern was to keep the segmentation result regardless of execution speed of each thread. The algorithm was built using the *OpenMP* library (Chapman et al., 2008) for programming with shared memory and was executed on a processor with four cores (quad-core). It was reached almost 2.6 in acceleration of the overall execution time.

The remainder of this paper is organized as follows. The next section provides a brief description of the region growing algorithm proposed by Baatz and Schäpe. In the following section, the proposed parallel implementation is described. In section 4, the results of an experimental analysis of performance are presented and section 5 concludes the work with the main conclusions and directions for future work.

2. SEGMENTATION BY REGION GROWING

This section briefly describes the sequential algorithm of region growing proposed by Baatz and Schäpe and used in the system Definiens (formerly eCognition) (Definiens, 2008).

The method is an iterative process of local optimization, which minimizes the average heterogeneity of the generated segments. The measure of heterogeneity used in the algorithm has a spatial component and a spectral component. The spectral heterogeneity is defined on the values of the spectral responses of the pixels contained in a segment. This measure is proportional to the weighted average standard deviation for each band.

Spatial heterogeneity is based on two shape attributes: smoothness and compactness. The degree of compactness is defined as the ratio between the perimeter of the segment and the square root of its area (number of pixels it contains). The smoothness is defined as the ratio between the perimeter of the object and the perimeter of the minimum boundary rectangle (bounding box).

Initially, each segment represents a single pixel of the image and all pixels are associated with a certain segment. The segments grow to the extent that they are united with their neighbours, and the smallest increase in heterogeneity is used as a criterion for selecting the neighbour with which a segment will be attached. To simulate a parallel growth, each segment is selected only once for each iteration.

The fusion factor (***f***) expresses the increase of heterogeneity resulting from the union of two segments. Before a union operation, the fusion factor is calculated for each of the neighbours of the selected segment. The neighbour which has the minimum fusion factor is chosen for merge. However, the union only occurs if the fusion factor is under certain threshold, defined as the square of the scale parameter, which will be denoted at this point of the text by the letter ***e***. This procedure continues merging segments until no more unions are possible.

The fusion factor contains a component for the spectral heterogeneity (***h_{color}***) and a component for the spatial heterogeneity (***h_{shape}***) (1). The relative importance of spatial and spectral components is defined by the color factor (***w_{color}***).

$$f = w_{color}.h_{color} + (1 - w_{color}).h_{shape} \quad (1)$$

Equation 2 shows the formulation of spectral heterogeneity; where the selected segment is ***obj1***, ***obj2*** is the analyzed neighbour and the ***obj3*** is the result of merge with ***obj2*** and ***obj1***. In this equation ***c*** is the index of the spectral band and ***w_c*** is an arbitrary weight set for band ***c***; ***σ*** is the standard deviation of the pixels in the band ***c***, considering all the pixels belonging to segment ***obji***; and ***n*** is the number of pixels in ***obji***, for ***i*** = 1,2,3.

$$h_{color} = \sum_i w_c (n_{obj3}.\sigma_c^{obj3}(n_{obj1}.\sigma_c^{obj1} - n_{obj2}.\sigma_c^{obj2})) \quad (2)$$

Spatial heterogeneity is influenced by the compactness degree of the segment and the smoothness of its edge (3). The measure of spatial heterogeneity, therefore, has two components: the component relative to compactness ***h_{cmpct}*** and the smoothness component ***$h_{smoothe}$***. The relative importance of these two components is defined by the factor of compression, ***w_{cmpct}***.

$$h_{shape} = w_{cmpct}.h_{cmpct} + (1 - w_{cmpct}).h_{smooth} \quad (3)$$

Equations 4 and 5 show the formulations of the components of compactness and smoothness. In these equations ***l*** is the perimeter of the segment ***obji*** and ***b*** the perimeter of the corresponding minimum bounding box for ***i*** = 1,2,3.

$$h_{cmpct} = n_{obj3}\cdot\frac{l_{obj3}}{\sqrt{n_{obj3}}} - (n_{obj1}\cdot\frac{l_{obj1}}{\sqrt{n_{obj1}}}) + n_{obj2}\cdot\frac{l_{obj2}}{\sqrt{n_{obj2}}}) \quad (4)$$

$$h_{smooth} = n_{obj3}\cdot\frac{l_{obj3}}{\sqrt{b_{obj3}}} - (n_{obj1}\cdot\frac{l_{obj1}}{\sqrt{b_{obj1}}}) + n_{obj2}\cdot\frac{l_{obj2}}{\sqrt{b_{obj2}}}) \quad (5)$$

The growth of the segments is constrained, therefore, an adjustable criteria of heterogeneity. This adjustment can be done by choosing the scale parameter (***e***), the weights of the spectral bands (***w_c***), the factor of color (***w_{color}***) and the compactness factor (***w_{cmpct}***). The changes on the scale parameter directly influence the size of the generated segments. Moreover, the relevance of each spectral band, the relative importance of shape and color, and between compactness and smoothness, can be adjusted through the parameters of the algorithm.

3. PARALLEL IMPLEMENTATION

The parallel implementation of the region growing algorithm proposed by Baatz and Schäpe uses the library *OpenMP* for parallelization and follows the division of computing in different threads that share the same data area in memory. The main idea of this solution consists in splitting the image into regions, that will be denoted tiles. Each tile is processed by a different thread, that perform a local region growing, using the sequential algorithm, with some synchronization actions.

This parallel approach, however, faces two major obstacles: (i) the treatment of boundary segments, i.e. segments that have at least one neighbour who does not belong to its tile, (ii) the reproducibility of the final result.

Regarding the treatment of the border segments of each tile, the main difficulty stems from the threads running in parallel. This can cause simultaneous treatment of the same segment for more than one thread. This could be avoided with the use of critical sections (zones where only one thread can execute at a time) to update the segment. Using critical sections, however, may cause great impact on segmentation performance, if the contention caused by waiting for critical sections is roughly the same as the gains from parallelization

In relation to the reproducibility of results, this is a problem inherent to the execution time of each thread. In other words, one thread can perform its task more quickly than others and may generate different orders of visitation for the segments, which affects the final result of segmentation. Even for sequential segmentation, if the seeds are visited in a different order, the result of the segmentation is modified. The reproducibility of the segmentation result, however, is an important goal, since it allows scientists from different locations to generate segmentations of the same image, and thus look at the same result.

To let the segmentation process be actually independent of the speed of the threads and to avoid excessive contention for critical sections, the segments located on the boundaries at the tiles are treated separately in the algorithm. These segments, called from now on frontier segments, are included in a list of segments to be treated. At the end of each step of segmentation (after all segments have been visited), the frontier segments will be processed sequentially. Therefore, the growth of regions of each thread will be independent, with no need for critical sections in the code.

The division of image in tiles, and consequently the division of work in threads, can impact the final result of segmentation. To achieve better performance in a given architecture with multiples cores, the ideal is that the number of threads is always equal to the number of processor cores available. In our implementation, the user defines the number of threads that will run on the processor. This guarantees the same tile division and the reproducibility of segmentation results for different architectures.

3.1 Initial Load Distribution

The first step of the algorithm consists in determining the number of tiles to be generated. The number of tiles corresponds to the amount of threads. If only a thread exists, the computation is sequential. For two or more threads, the image is divided into distinct areas, as shown in Figure 1. Each thread is responsible for processing the pixels included in its tile.

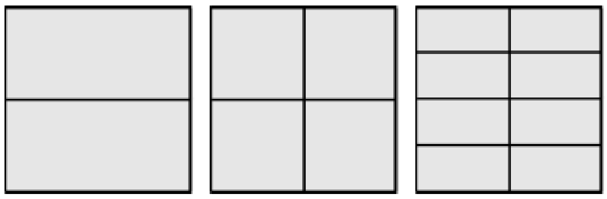
Figure 1. Division into tiles for two, four and eight threads

3.2 Growth Step

After the initial division of the image, begins the growth stage begins. Each thread executes the region growing algorithm inside its own tile. Initially, the thread marks all pixels of the tile as segments to be visited and organizes them into a list of segments. The segments are included in this list in the same order that they will be visited. In order to start the growth from relatively distant segments, the segments are included according to their relative distance in the image.

A thread visits each segment on its list, and analyzes the heterogeneity increases for each of its neighbours. If at least one of the neighbours does not belong to the tile treated by that thread, the segment is included in the list of frontier segments. Otherwise, the segment is processed normally and marked as visited. The neighbour that results in the less heterogeneity increase is considered the best neighbour. If this best neighbour is considered, by the fusion factor, as part of the segment, then a merge occurs. This procedure is repeated until the entire list of segments in each thread is covered. Figure 2 gives an example of a segmentation divided into four tiles, showing on green the frontier segments.

After all the threads finish their computation, the frontier segments are handled. The list of frontier segments is traversed sequentially by a single thread, using the same region growing algorithm. The frontier segment list is visited in an interleaving fashion, one frontier segment from each tile at time. Thus, the segments remain visited in a distributed way.

After the list of frontier segments is processed, the growth stage is completed, and the algorithm starts a new step, with another growth stage. The new growth stage starts in the same way, with a number of threads computing the segments of each tile. In this stage, however, the list of segments of each thread is composed by the segments generated in the previous step. This process is repeated, generating new steps, until no merge occurs in a step or until a maximum number of steps is reached.

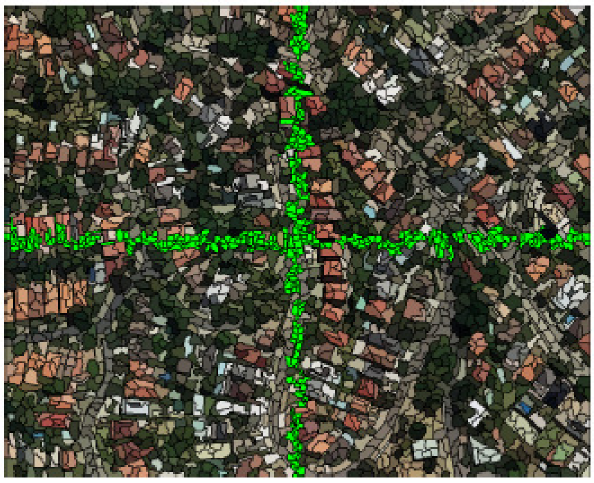
Figure 2. Frontier segments on green in four tiles

4. RESULTS

In this section, the results obtained with the parallel implementation of the region growing segmentation are presented. The following sections describe the environment used in the experiments, the test images, and the segmentation results, along with the evaluation of the performance obtained with the parallelization.

4.1 Test Environment

The experiments were all performed on an Intel Core 2 Quad 2.40 GHz, 2 GB of RAM.

Three images with different sizes and features were used. They are named as Im1, Im2 and Im3 and exposed, respectively, in Figures 2, 3 and 4. Image sizes are presented in Table 1. All images were used to evaluate the performance gains and also used to compare the result generated from parallel segmentation to the result from sequential segmentation.

Image	Size (pixels)
Im1	1000 x 1000
Im2	2000 x 2000
Im3	2800 x 2800

Table 1. Images used for experiments and its sizes

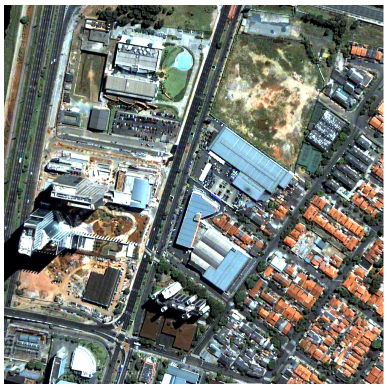
Figure 2. Im1

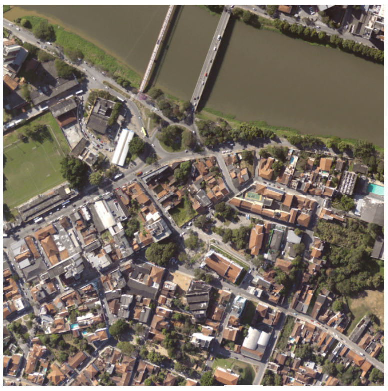
Figure 3. Im2

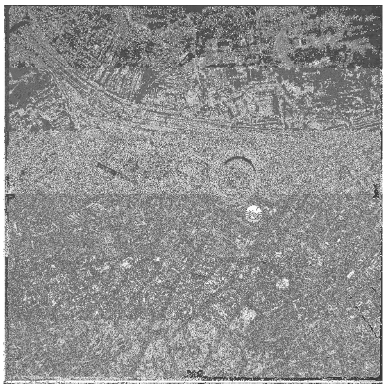
Figure 4. Im3

4.2 Results of Parallel Segmentation

Figures 6, 8 and 10 show the result of the sequential segmentation for Im1, Im2 and Im3. Figures 7, 9 and 11 expose the results of parallel processing of the same images.

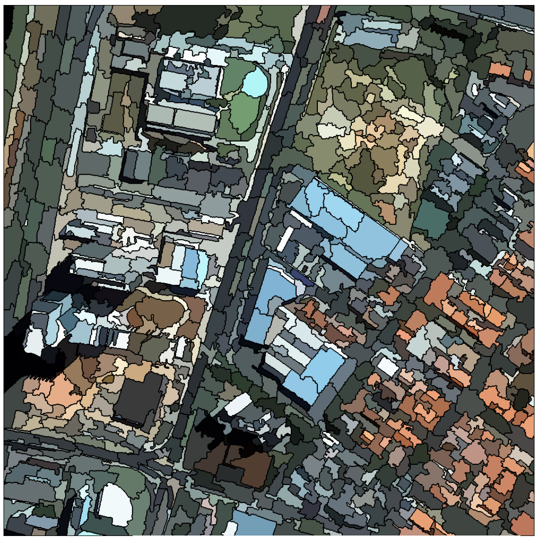
Figure 6. Result of sequential segmentation for Im1

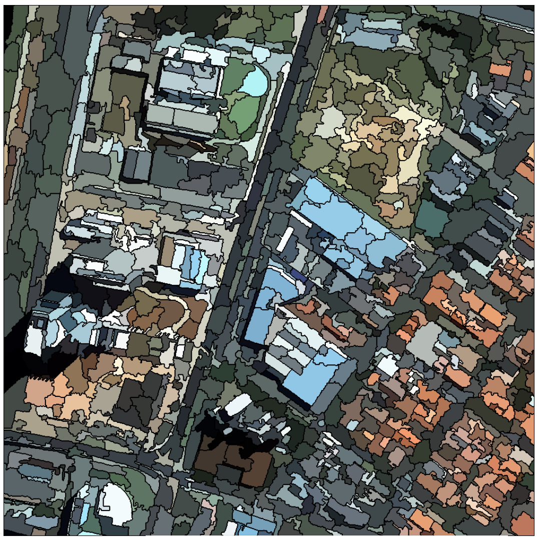
Figure 7. Result of paralell segmentation for Im1

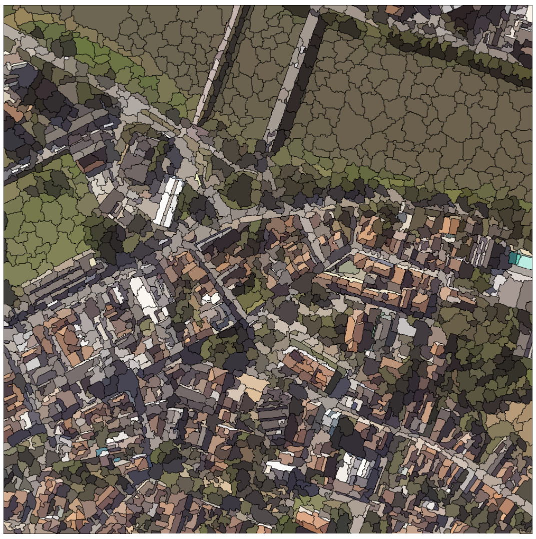
Figure 8. Result of sequential segmentation for Im2

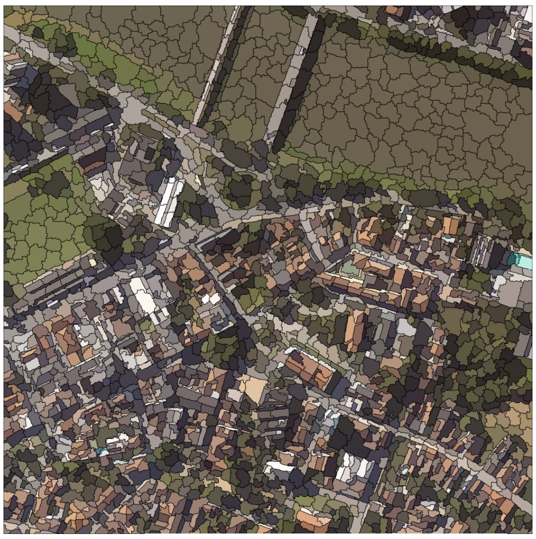
Figure 9. Result of parallel segmentation for Im2

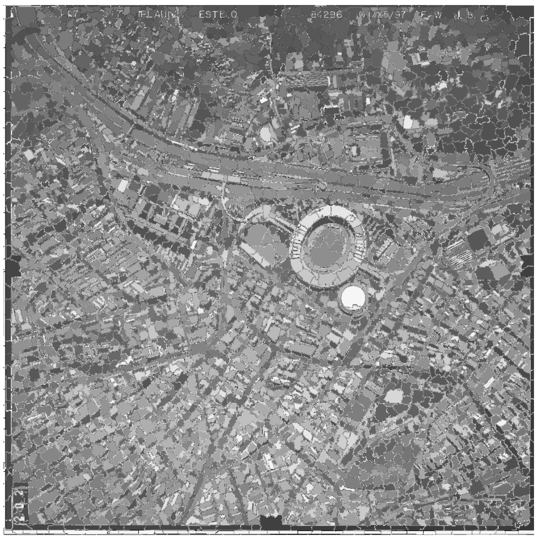
Figure 10. Result of sequential segmentation for Im2

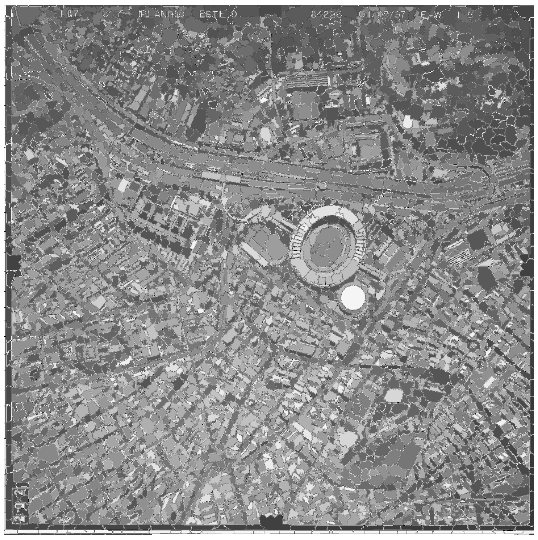
Figure 11. Result of parallel segmentation for Im2

It can be noted in the Figures the similarity between the results of parallel and sequential segmentation. There are also no important differences in the central region, the most challenging area, where the segments belonging to the boundary of the tiles are. The divergences that can be observed in the segmentation results are due solely to the difference in the order in which segments are processed, which could occur even in a sequential algorithm if this order is modified. It is important to notice that for the same number of threads, all the segmentation results are exactly the same no matter how many times the parallel algorithm runs.

4.3 Performance Evaluation

The performance of the proposed parallel algorithm has been evaluated using the same three images and varying the number of threads. Table 2 presents the execution time of segmentation against the number of threads executed. Note that execution time of the segmentation is reduced with the increase in the number of threads.

Image	Average Time (seconds)		
	1 thread	2 threads	4 threads
Im1	13.67	8	6.33
Im2	62.33	39.33	24
Im3	123.66	76	51

Table 2. Time elapsed for each segmentation process

The results suggest an even greater reduction of segmentation time if more processors are used. Obviously, in the environment tested, does not compensate to run a number of threads greater than the number of processor cores. However, nowadays, many high-performance systems present two or more multicore processors sharing the same memory. The algorithm could benefit from this type of architecture, reducing even more the execution time.

Table 3 shows the speedup obtained by the suggested parallel algorithm. The speedup is measured as the ratio between the time of sequential execution and parallel execution time and shows the relative increase of the parallel performance.

Image	Average Time (seconds)		
	1 thread	2 threads	4 threads
Im1	1	1.70	2.16
Im2	1	1.58	2.59
Im3	1	1.63	2.42

Table 3. Speedup obtained by parallel algorithm

As noted in this table, speedups of up to 2.59 were obtained. The total utilization of four processor cores was not possible due to the inherently sequential part of the algorithm. This part is required to maintain the reproducibility of results. Nevertheless, for very large images, reducing the segmentation time in 2.5 times is an important result, considering that the algorithm explores the full potential of the hardware present in most of the current desktop computers.

In terms of the different images testes, it can be observed that the segmentation of Im1 presented the greatest speedup for two threads, but was the least benefited from the increase of 2 to 4 threads. In the other hand, Im2 segmentation achieved the smallest speedup for two threads, but obtained the greatest increase, reaching a speedup of almost 2.6 for 4 threads. It can be observed that the speedups obtained are not only influenced by the image sizes, but also by the spectral and spatial attributes of image.

Another test was executed by varying some segmentation parameters. Table 4 presents the time and the speedup achieved when changing the color weight ($\boldsymbol{w_{color}}$) from 0.9 to 0.1, i.e., prioritizing shape to color.

Image	Average Time (seconds)		Speedup
	1 thread	4 threads	4 threads
Im1	17	7	2.42
Im2	72.33	29	2.49
Im3	143.33	55.67	2.57

Table 4. Time and Speedup obtained when $\boldsymbol{w_{color}} = 0.1$

Comparing the results presented on Table 3 with the values on Table 4 it can be observed that the speedup obtained for Im1 and Im3 were better when the shape was prioritized, but the same did not happen for Im2. This result reinforces that the image features influences on the segmentation product.

5. CONCLUSIONS

This work presents a parallel region growing algorithm based on the algorithm proposed by Baatz and Schäpe. It was developed an algorithm using *OpenMP* threads in order to leverage the parallel processing capability of current processors with multiple cores.

The focus of this implementation was to improve the performance of segmentation, keeping the reproducibility of results. The computation is divided by tiles and the frontier segments are processed sequentially. In terms of performance, parallel implementation was about two and a half times faster than the sequential segmentation. This is a very promising result since it allows the exploitation of the vast processing power of current processors with multiple cores.

In the future, the intention is to use the same principle of division of work in tiles to write an out-of-core version of the segmentation algorithm. This version would allow the segmentation of images that do not fit in main memory. Thus, it is expected that the image segmentation can handle extremely large data efficiently and without requiring special hardware. It is also expected to propose others parallel versions for different hardware like clusters and GPUs (Graphics Processing Units).

6. REFERENCES

Baatz, M. and Schäpe, A. “Multiresolution segmentation: an optimization approach for high quality multi-scale image segmentation”. In: XII Angewandte Geographische Informationsverarbeitung, Wichmann-Verlag, Heidelberg, 2000.

Blaschke, T. and Strobl, J. “What is wrong with pixels? Some recent developments interfacing remote sensing and GIS”. GIS-Zeitschrift für eoinformationssysteme, 2001.

Chapman, B., Jost, G., van der Pas, R. Using OpenMP: Portable Shared Memory Parallel Programming. The MIT Press, 2008.

DEFINIENS, Image Analysis Software for Earth Sciences, http://www.definiens.com/image-analysis-for-earthsciences_45_7_9.html (último acesso 14 Novembro 2008).

Pal, N. R. and Pal, S. K., “A review of image segmentation techniques”, Pattern Recognition, 26(9):1277-94, 1993.

Wassenberg, J., Middelmann, W. and Sanders, P., "An Efficient Parallel Algorithm for Graph-Based Image Segmentation", CAIP '09: Proceedings of the 13th International Conference on Computer Analysis of Images and Patterns, 2009.