International Journal of
Applied Mathematics,
Electronics and Computers
ISSN: 2147-8228 http://ijamec.atscience.org

Advanced Technology and Science

Original Research Paper

# Parallelization of a Hierarchical Graph-Based Image Segmentation using OpenMP

**Ali Saglam *[1], Nurdan Akhan Baykan [1]**

*Accepted 3rd September 2016*

***Abstract:*** In many image-processing applications, image segmentation is an essential stage. In this stage, an image is partitioned into several regions according to the similarity of its pixels. In addition to the accuracy of the image segmentation, the speed is also very important for real-time image processing applications. Many computer applications take advantages of the multi-processor architecture to up to their running performance. However, to run an algorithm as parallel is very difficult in many cases. Due to using the same memory blocks, many conflicts might be happened between the processors. Moreover, each process of one processor may depend on those of another processor. For this reason, the algorithm to be parallelized must be suitable to parallel. In addition, the processing traffic that is pursued by the processors must be controlled within some parallel directives. In this paper, we provide a parallel implementation to a hierarchical graph-based image segmentation method by using its hierarchical processing steps. To achieve this goal, we utilize the OpenMP (Open Multi-Processing) Library to run the segmentation process as parallel on images of different sizes from the INRIA Holidays dataset. The experimental results show that the parallel implementation of the algorithm is more effective than the serial type according to processing time.

***Keywords:*** *Parallel programming, Image segmentation, Graph, OpenMP.*

## 1. Introduction

For many image processing application, the image segmentation is one of the low-level and mid-level image processing stages. During the image segmentation stage, the pixels of an image are grouped into regions according to the similarity in the regions and differences among the regions. This stage is one of the most difficult stages in the image processing applications [1]. The consequence of this stage affects all of the following stages and their results. Image segmentation reduces the large amounts of data for the analysis operations. In an image, the segmentation process inherently focuses on the homogeneous regions and the borders between them. It performs this task by using brightness, color, pattern (texture), or density differences, etc. [2,3].

Many pattern recognition and computer vision applications take advantage of the global features of the images for more accurate segmentation. However, in this way, the processing time exceeds the allocated time for the input frame periods of the used camera in many real-time computer visions system. Due to the computational complexity of the methods that take into account the global features of the images, those that consider local features are mostly preferred in real-time applications [2].

Graph-based methods are some of the segmentation methods. Because of their representation relevance and ease of retention of images, the graph theory tools are utilized as prominent tools in computer vision [2]. In graph-based segmentation methods, the nodes (or vertices) of the graph refer the pixels of the image and the edges define the connections between the adjacent (or neighboring) vertices. Each of these edges generally has a numerical value, which is called weight [4]. The weights of the edges are considered the difference of the gray level values or color values (if the segmentation is performed on a color image) of the vertices that exists at the ends of the edge. The difference is generally computed by using Euclidean distance measurement [5]. Spanning trees are sub-graphs of a graph; such that they do not contain any loop and it is easy to make process on the graph via the spanning tree structure. The spanning tree of a graph that has the least total edge weight among the other spanning trees of the graph is called minimum spanning tree (MST) [4]. Image segmentation algorithms based on minimum spanning tree take advantage of the features of the MST, because it takes no account of most of the edges on the image-graph. Therefore, the cost of the operation is reduced significantly [6].

The segmentation process in MST-based segmentation algorithms comes true in two ways. The first way is the cutting procedure in which the edges to be cut are removed on the MST that covers all of the pixels of the image. The edges to be cut represent the borders on the image. By this way, a tree is separated into several sub-trees. The sub-trees represent regions in the image [5,7-9]. The second way is the merging procedure in which the sub-trees that satisfies a homogeneity criterion are merged [10]. If two sub-trees do not satisfy the criterion, they are not merged, and in this case, the border between the two regions that the two sub-trees represent occurs. In this paper, we parallelize the merging procedure that is performed hierarchically.

Efficient Graph-Based Image Segmentation algorithm, which is proposed by Felzenszwalb and Huttenlocher, defines a merging criterion [10]. The algorithm employs the Kruskal's algorithm [11], which is one of the most popular MST algorithms. In this algorithm, all of the edges of the graph structured an image is sorted in ascending order according to their weight. Then, the edges are added to the MST structure by starting the shortest edge, such that the MST is empty at first. If the trees that are at the ends of the edge to be added do not satisfy the merging criterion, the edge is not added. By the way, the MST may be divided into more than one sub-trees. Haxhimusa and Kropatsch [12] used this merging criterion on Boruvka's hierarchical MST algorithm [13]. In this algorithm, each node (vertex) assumed a sub-tree, and each merging process of these sub-trees actualizes independently from others. For this reason, this makes it suitable for parallelization.

[1] *Selcuk University Konya/Turkey*
** Corresponding Author: Email: alisaglam666@gmail.com*
*Note: This paper has been presented at the 3rd International Conference on Advanced Technology & Sciences (ICAT'16) held in Konya (Turkey), September 01-03, 2016.*

(a) (b) (c)

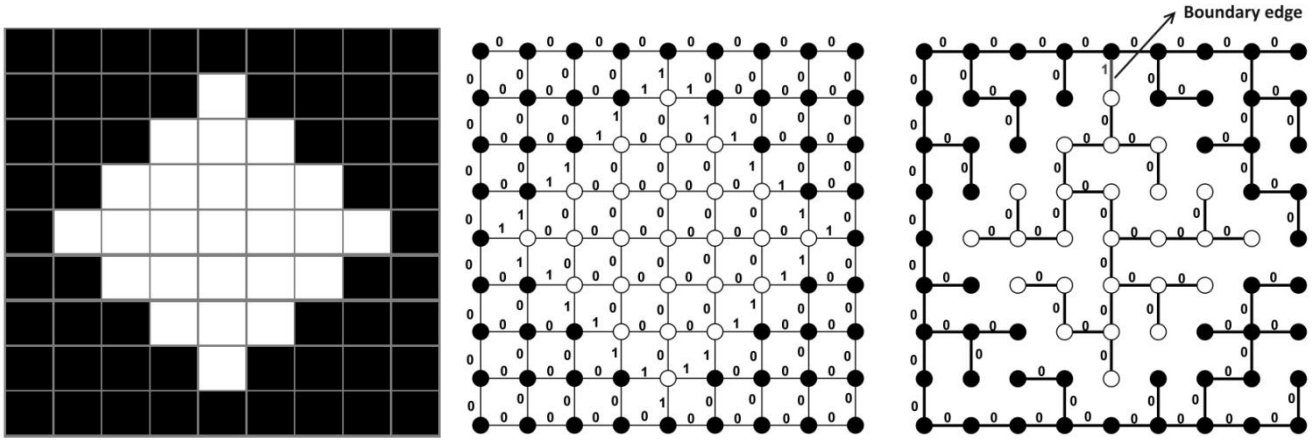


**Figure 1.** (a) A simple black and white image , (b) the graph representation of the image according to 4-connection neighborhood, (c) the MST of the image extracted from its graph representation and an example of boundary edge in the MST

Nowadays, the multi-core processors are prevalent; and therefore, the parallel programming has been a very popular issue. Scientific and engineering programs such as image-processing applications need to be fast to overcome the heavy processing load in limited times. In order to make use of this technology, parallelization of the applications reduces the process load. Obtaining the same or similar results with sequential execution and ensuring to not overlapping of the parallel operations is mostly important and difficult matters for parallel programming issue. In this paper, we organize the hierarchical graph-based image segmentation algorithm as appropriate for parallel execution [14].

## 2. The Hierarchical Graph-Based Image Segmentation

Let the graph $G = (V, E)$ be the image-graph that is constructed from an image. $V$ represents the set of the vertices in the graph $G$, such that each vertex refers a pixel in the image and $V$ contains all of the pixel-vertices. $E$ represents the set of the edges that connects each vertex to its neighboring vertices. In this paper, 8-connected neighborhood is used as a neighborhood system. The edges are generally weighted and undirected. The weight of each edge is a numerical and positive value that indicates the degree of similarity between two vertices at the ends of the edge. As the degree of similarity two vertices is lower, the weight value is higher. To figure out the similarity, the one-dimensional gray-scale values or multi-dimensional color values of the pixels can be used, and to compute the difference for both ways, the Euclidean distance measurement can be used. The edge $(u, v)$ connects the vertices $u$ and $v$, and $w(u, v)$ is the weight value of the edge. Let $x_u$ be the one-level gray-scale value, and $x_{u,i}$ be the $i$th value of the $z$-level color vector of the vertex $u$ where $i = \{1,2,\cdots,z\}$. The Euclidean distance as the weight value of $(u, v)$ is computed as in Eq. 1.

$$w(u,v) = \left(\sum_{i=1}^{z} \left(x_{u,i} - x_{v,i}\right)^2\right)^{1/2} \quad (1)$$

### 2.1. Boruvka's Minimum Spanning Tree Algorithm

Boruvka's MST algorithm constructs an MST structure from a graph, such that the graph must be a weighted, undirected, and complete graph (Fig. 1). According to the algorithm, each vertex is assumed as a tree at first stage and no one is connected to any other one. The union of multiple trees is called forest. At first, all of the vertices compose the forest $F_0$, because each vertex is a tree such that $F_0 = T_1 \cup T_2 \cup \cdots \cup T_n$, such that n refers to the number of the vertices on the graph $G$. $T_i$ refers to the $i$th tree ($T_i$ equals to the $i$th vertex at first stage) and each tree $T_i$ chooses the smallest edge $(u, v)$ where $u \in T_i$ and $v \notin T_i$, then the edge is added to the new forest $F_1$ that is being composed for the next stage. At the begin of the next stage, the number of the trees will have been reduced by at least one-half, i.e., will have been decreased by at least one-half. If any edge is not added in the current stage, on the other words, the last forest $F_{last}$ has one tree, the algorithm ends.

### 2.2. The Graph-Based Segmentation Algorithm

In the MST of a graph representation of an image, the highest weighted edges represent the boundaries of the image (Fig. 1). Felzenszwalb and Huttenlocher have segmented the images by detecting the edges of the boundaries, which are called boundary edges [10,15]. For the purpose of to detect the boundary edges to be excluded from the MST, Felzenszwalb and Huttenlocher have used the merging criterion Eq. 2 in the Kruskal's MST algorithm. According to the Kruskal's algorithm, each vertex in the graph $G$ that is representation of the image seems as a tree. Firstly, all of the edges in $G$ are sorted in order to their weight in ascending order. The two trees at the ends of the smallest weight edge in the sorted edge queue are merged if the two trees are not the same tree, and the edge is removed from the queue. At last, only one single tree remains such that it is called MST [11]. If the merging criterion in Eq. 2 is applied on the algorithm, more than one MSTs might remain at the end of this process [10], because the edges that does not satisfy the criterion are not added to the MST to be formed, and thus, the MST that covers all of the vertices might not be created.

$$Diff(T_1, T_2) < min\left(Int(T_1) + \frac{k}{|T_1|}, Int(T_2) + \frac{k}{|T_2|}\right) \quad (2)$$

In Eq. 2, $T_1$ and $T_2$ are the sub-trees to be merged. $Diff(T_1, T_2)$ defines the weight value of the shortest weighted edge that connects the trees $T_1$ and $T_2$. $Int(T_1)$ and $Int(T_2)$ defines the internal difference of $T_1$ and $T_2$ respectively. An internal difference of a tree is the weight value of the highest weighted edge in the tree. $T_1$ and $T_2$ defines the number of vertices in the trees $T_1$ and $T_2$ respectively. The constant value $k$ refers to a threshold value that controls how much greater $Diff(T_1, T_2)$ must be than the internal differences of the trees $T_1$ and $T_2$.

In this paper, Eq. 2 is utilized as a merging criterion in the Boruvka's algorithm, because the Boruvka's algorithm progresses

**Input:** Given image graph $G = (V, E)$
**Output:** Tree set $T = T_1, T_2, \ldots, T_t$

**for** $i \leftarrow 1$ **to** $|V|$ **do**
  $T_i \leftarrow v_i$
**end for**
$counter \leftarrow |V|$
**repeat**
  $t \leftarrow counter$
  **for** $i \leftarrow 1$ **to** $t$ **do**
    $(u, v) \leftarrow T_i.smallest\_adjoining\_edge$
    **if** $w(u, v) \geq \min\left(Int\big(T(u)\big) + \frac{k}{|T(u)|}, Int\big(T(v)\big) + \frac{k}{|T(v)|}\right)$
      $continue$
    **end if**
    $merge\ T_u\ and\ T_v$
    $counter \leftarrow counter - 1$
  **end for**
**until** $counter = t$

**Figure 2.** The steps of the hierarchical segmentation algorithm.

in a hierarchical order. Due to this reason, this algorithm is more suitable for parallelization than the other popular MST algorithms [16]. The hierarchical graph-based image segmentation algorithm is sketched in Fig. 2.

In Fig. 2, the constructed graph $G = (V, E)$ is obtained from the image used, and then it is taken to the process as an input data. Firstly, each vertex seems as a tree. The number of trees $t$ is equal to the number of vertices $V$ initially. For each tree $T_i$, the smallest edge $(u, v)$, where $u \in T_i$ and $v \notin T_i$, is found. If the weight value $w(u, v)$ satisfies the criterion in Eq. 2, the tree $T_u$ to which the vertex $u$ belongs and the tree $T_v$ to which the vertex $v$ belongs are merged, and thus, the number of tree is reduced to one. After these processes are complete for each tree, this cycle is repeated iteratively. If no merging is happened, the algorithm ends.

## 3. The Parallel Implementation of The Algorithm

### 3.1. OpenMP

Multi-core processors have become quite prevalent nowadays. Scientific and engineering programs such as image-processing applications need to be fast to overcome the high processing load in limited times. In order to make use of the multi-core processor technology, such applications need to be parallelized. Therefore, some libraries and tools, which are developed to take advantage of the multi-core processor architecture, should be used and the programs that will be parallelized must be coded suitable for parallelization. OpenMP (Open Multi-Processing) is a run-time program library that has routines and environment variables and includes a set of compiler directives. The directives can be embedded into C/C++ and Fortran programming languages using shared-memory platforms to run codes on more processors [17]. On the platforms like these, OpenMP provides an easier programming model. However, getting high performance from OpenMP sometimes might be a very difficult task. The acceleration rate of the program that use OpenMP directives depends on many factors, including the problem how the variables are used in the code, data layout, workload balancing and so on. OpenMP is especially useful for partitioning a loop and running the iterations of the loop on more processors simultaneously. For that, the loop dependencies need to be configured in accordance to parallelism [17].

### 3.2. Implementation of The Parallel Directives

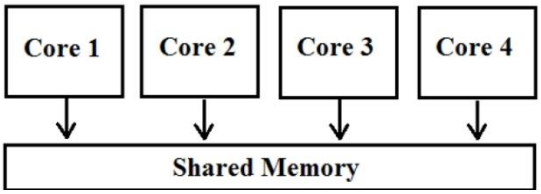


**Figure 3.** A simple presentation of a shared memory architecture.

The parallel computation is put into practice in three ways: general-purpose computing on graphics processing units, via a message processing interface, or using shared memory architecture (OpenMP) [18]. In a shared memory model system, threads, which are processes such that each runs as parallel, are communicated each other by reading and writing directly to the same memory. Since the OpenMP specification was introduced as a standard shared-memory programming model in 1998, there have been many studies to implement the OpenMP directives to their programs to run as parallel [19]. By using the OpenMP instructions, it is easy to solve the high processing load problems of the classic single pipeline program model, because OpenMP execute some calibrations automatically, and we do not need to consider which core should run in which part of the calculation [20].

OpenMP uses a fork-join model, which is a standard way to execute a parallel program on shared-memory systems (Fig. 3). Firstly, a parallel program begins to run on a single thread called master thread. Once the process flow encounters a parallel block, the process flow is separated to several threads (known as the fork) to run on additional processor cores [21].

OpenMP provides several mechanisms to run the code as parallel. Additionally, OpenMP provides an extra translator that analyzes the parallel loop and generate executor code before the OpenMP translation [19]. The translator determines the number of cores to run, separates the loop into sub-loops automatically according to the number of the cores, and employs each sub-loops to different threats. The example of its use is as below:

```
#pragma omp parallel for
for(i=0;i<1000;i++)
{
   x[i] = y[i];
}
```

OpenMP provides some directives as *private* and *shared* to define how the parallel regions use the data values. In parallel programming, synchronization is used to impose some order constraints and to organize to access to shared data. OpenMP also provides several mechanisms to synchronize the parallel threads, with *critical*, *atomic*, *lock*, *ordered*, *flush*, and *barrier* [14,22].

To synchronize the parallel graph-based algorithm, we used the *lock* directive. In the block for merging clusters, if we embed the locking mechanism in the processes, the threads that each works on a different cluster will also wait to each other [23]. In this way, the flow would not be parallel. Therefore, we restricted to the threads that own the same index of lock array so that the index is the cluster number at the same time. The example of its use is as below:

```
mylock = new omp_lock_t();
omp_init_lock(&(mylock));

lock_array = new omp_lock_t[size];

for (i = 0; i < size; i++)
{
   omp_init_lock(&(lock_array[i]));
}
```

If a lock is set with the expression *omp_set_lock* by a processor, other processors that wants to set this lock must be waiting the processor until the lock is unset with the expression *omp_unset_lock* by the processor that has set the lock.
Another problem for synchronizing the processes is the mutual exclusion problem. In cases when two clusters chose each other as nearest cluster on different threads simultaneously, a crash may happen. To solve this problem, we get the related code into a critical block by the directive *critical*. By this way, only one processor employs the codes that are in the critical block as seen below:

```
#pragma_omp_critical
{
   omp_set_lock(mylock1);
   omp_set_lock(mylock2);
}
```

There is also another directive for getting the code into a critical block, which is called *atomic* such that it is used for only one variable to perform faster processing [17]. For example:

```
#pragma_omp_atomic
   a = a - 1;
```

The parallel implemented version of the algorithm in Fig. 2 is sketched in Fig. 4.

## 4. Experimental Results and Discussion

The system that has been used for the values in Table I has 32 bit Intel i5-2400 3.10-GHz Quad Core processor with 4 GB RAM. We used C++ programming language. To import and illustrate the images, we used OpenCV (Open Source Computer Vision) library [24].

**Input:** Given image graph $G = (V, E)$
**Output:** Tree set $T = T_1, T_2, \dots, T_t$

lock_array = ***new omp_lock_t***[|$V$|]
**for** $i \leftarrow 1$ **to**|$V$| **do**
  $T_i \leftarrow v_i$
  ***omp_init_lock***(&(lock_array[$i$]))
**end for**
$counter \leftarrow |V|$
**repeat**
  $t \leftarrow counter$
  **#pragma omp parallel for private**($i$) **shared**($counter$)
  **for** $i \leftarrow 1$ **to** $t$ **do**
    $(u, v) \leftarrow T_i.smallest\_adjoining\_edge$
    **#pragma omp critical**
    {
      **omp_set_lock**(&(lock_array[$T(u).id$]));
      **omp_set_lock**(&(lock_array[$T(v).id$]));
    }
    **if** $w(u,v) \geq \min\left(Int\big(T(u)\big) + \frac{k}{|T(u)|}, Int\big(T(v)\big) + \frac{k}{|T(v)|}\right)$
      **omp_unset_lock**(&(lock_array [$T(u).id$]));
      **omp_unset_lock**(&(lock_array [$T(v).id$]));
      $continue$
    **end if**
    $merge\ T_u\ and\ T_v$
    **#pragma omp atomic**
      $counter \leftarrow counter - 1$
    **omp_unset_lock**(&(lock_array [$T(u).id$]));
    **omp_unset_lock**(&(lock_array [$T(v).id$]));
  **end for**
**until** $counter = t$

**Figure 4.** Parallel implementation of the algorithm in Fig. 2.

In Fig. 5, the original versions of some images from INRIA Holidays dataset [25], their segmentation results and which processor has performed on which segments of the images in last iteration are presented.
We applied the algorithm on some images from INRIA Holidays dataset [25] in the different versions of the images as the sizes of 128×96, 256×192, 512×384, and 1024×768. We executed the program five times for each elapsed time value in Table 1, and put the smallest one of the five values to the table. The time unit is given as seconds (s) in Table 1.

## 5. Conclusion

As seen in the results, parallel programming provides an important acceleration in processing speed. However, the processing time

**Table 1.** The elapsed times (s) of some images from the dataset with the different image sizes and the number of cores

| Image Sizes | 128×96 | | | | 256 × 192 | | | | 512 ×384 | | | | 1024 ×768 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Number of Cores | | | | | | | | | | | | | | | |
| Image Names | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 |
| 200200 | 0.036 | 0.031 | 0.028 | 0.029 | 0.172 | 0.135 | 0.123 | 0.123 | 0.745 | 0.583 | 0.543 | 0.518 | 3.171 | 2.427 | 2.203 | 2.133 |
| 203600 | 0.036 | 0.030 | 0.028 | 0.028 | 0.179 | 0.138 | 0.127 | 0.123 | 0.781 | 0.602 | 0.550 | 0.526 | 3.308 | 2.492 | 2.317 | 2.168 |
| 203900 | 0.035 | 0.029 | 0.027 | 0.029 | 0.179 | 0.137 | 0.128 | 0.123 | 0.776 | 0.572 | 0.519 | 0.510 | 3.074 | 2.337 | 2.118 | 2.099 |
| 206400 | 0.038 | 0.031 | 0.029 | 0.029 | 0.179 | 0.140 | 0.129 | 0.126 | 0.753 | 0.581 | 0.536 | 0.524 | 3.200 | 2.460 | 2.310 | 2.195 |
| 207000 | 0.037 | 0.031 | 0.029 | 0.031 | 0.178 | 0.138 | 0.131 | 0.127 | 0.773 | 0.576 | 0.525 | 0.544 | 3.157 | 2.407 | 2.233 | 2.143 |
| 207600 | 0.038 | 0.031 | 0.029 | 0.029 | 0.185 | 0.150 | 0.132 | 0.129 | 0.754 | 0.594 | 0.541 | 0.536 | 3.162 | 2.402 | 2.272 | 2.154 |
| 211100 | 0.036 | 0.031 | 0.029 | 0.029 | 0.186 | 0.146 | 0.132 | 0.131 | 0.723 | 0.584 | 0.519 | 0.535 | 3.152 | 2.408 | 2.210 | 2.154 |
| 213700 | 0.036 | 0.031 | 0.029 | 0.030 | 0.180 | 0.144 | 0.131 | 0.132 | 0.756 | 0.593 | 0.539 | 0.533 | 3.196 | 2.481 | 2.221 | 2.209 |

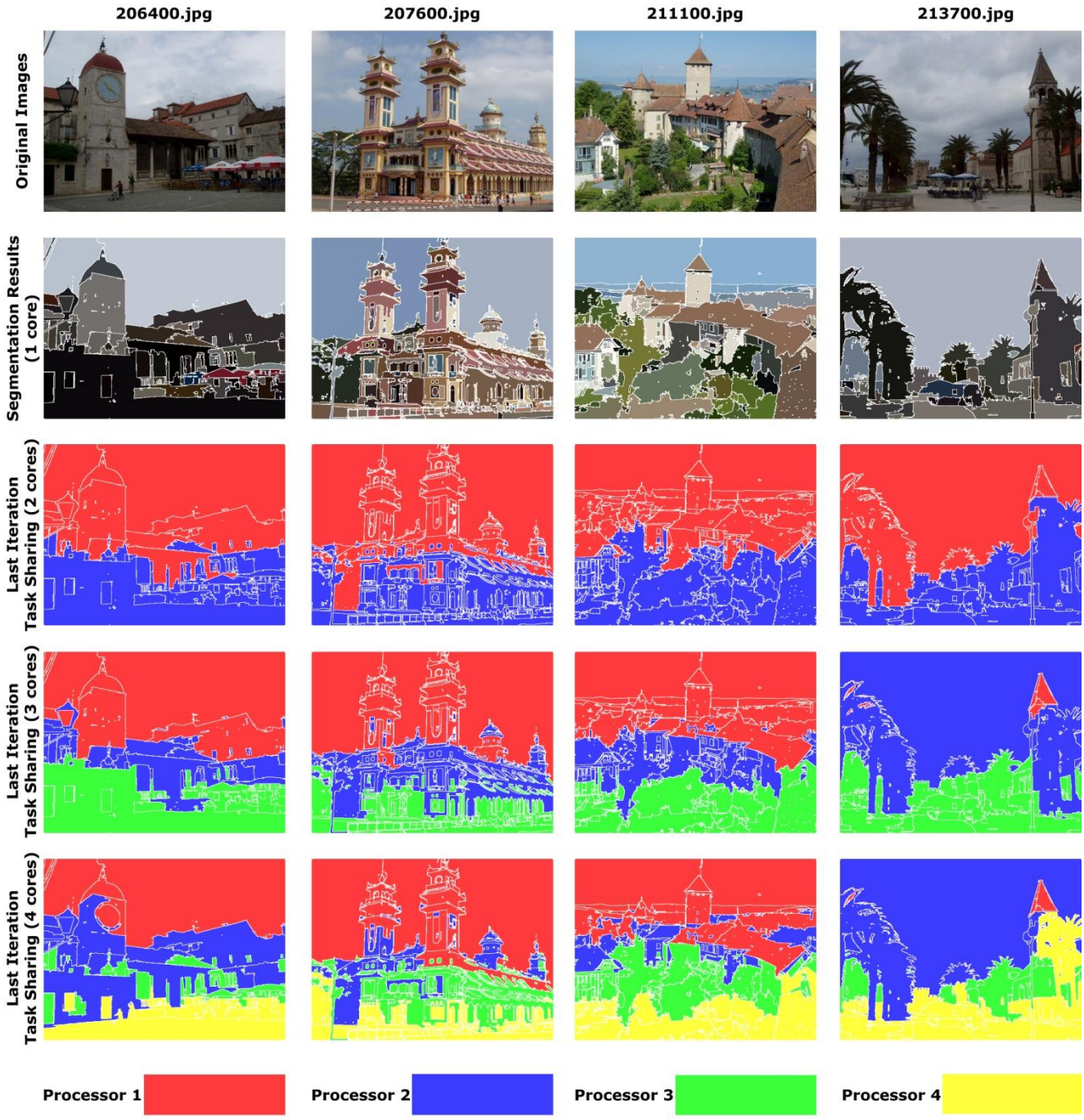


**Figure 5.** Visualization of the segmentation results and the task sharing of the processors for some images.

decreases no more in all cases, as the number of cores increases. The speedup of the execution relies on many factors, including compiler optimizations, runtime support, data layout, operating system noise, workload balancing and so on. Additionally, it may be depend on the regions to merge in the structure of the graph of the image. In a parallel loop, even if the threads complete their process except one, they will have to wait for it. In these cases, the operating system might have executed another task on the thread in the meantime. Nevertheless, the parallelization saved a noticeable time for us.

## Acknowledgements

This study has been presented as an oral presentation at ICAT'16 conference held in Konya (Turkey), 1-3 September 2016 and selected for the journal of IJAMEC (International Journal of Applied Mathematics, Electronics and Computers).

## References

[1] R. Nikhil and K. Scansar, "A review on image segmentation techniques," Pattern Recognition, vol. 26, no. 9. pp. 1277–1294, 1993.

[2] B. Peng, L. Zhang, and D. Zhang, "A survey of graph theoretical approaches to image segmentation," Pattern Recognit., vol. 46, no. 3, pp. 1020–1038, 2013.

[3] W. Tao, H. Jin, and Y. Zhang, "Color image segmentation based on mean shift and normalized cuts.," IEEE Trans. Syst.

Man. Cybern. B. Cybern., vol. 37, no. 5, pp. 1382–1389, 2007.

[4] J. A. Bondy, Graph Theory With Applications. Oxford, UK, UK: Elsevier Science Ltd., 1976.

[5] C. T. Zahn, "Graph-Theoretical Methods for Detecting and Describing Gestalt Clusters," IEEE Trans. Comput., vol. C$-$20, no. 1, pp. 68–86, 1971.

[6] O. J. Morris, M. D. J. Lee, and A. G. Constantinides, "Graph Theory for Image Analysis : An Approach Based on The Shortest Spanning Tree," Commun. Radar Signal Process. IEE Proc. F, vol. 133, no. 2, pp. 146–152, 1986.

[7] Y. X. V. Olman, D. Xu, "Solving data clustering problem as a string search problem," in Proc. Conf. Stat. Data Mining Know. Dis., Chapman \& Hall CRC, 2003, pp. 417–434.

[8] J. Shi and J. Malik, "Normalized Cuts and Image Segmentation," IEEE Trans. Pattern Anal. Mach. Intell., vol. 22, no. 8, pp. 888–905, 2000.

[9] A. Saglam and N. A. Baykan, "Sequential image segmentation based on minimum spanning tree representation," Pattern Recognit. Lett., Available online 16 June 2016, ISSN 0167-8655, http://dx.doi.org/10.1016/j.patrec.2016.06.001.

[10] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient graph-based image segmentation," Int. J. Comput. Vis., vol. 59, no. 2, pp. 167–181, 2004.

[11] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," Proc. Am. Math. Soc., vol. 7, no. 1, pp. 48–48, 1956.

[12] Y. Haxhimusa and W. Kropatsch, "Segmentation graph hierarchies," Lect. Notes Comput. Sci. 3138, pp. 343–351, 2004.

[13] O. Borůvka, "O jistém problému minimálním," Práce Morav. přírodovědecké společnosti, no. 3, pp. 37–58, 1926.

[14] A. Marongiu, P. Burgio, and L. Benini, "Supporting OpenMP on a multi-cluster embedded MPSoC," in Microprocessors and Microsystems, 2011, vol. 35, no. 8, pp. 668–682.

[15] A. Saglam, Minimum Yayilan Agac Tabanli Sirali Goruntu Bolutleme (Minimum Spanning Tree-based Sequential Image Segmentation), Master thesis, Selcuk University, Turkey, 2016.

[16] M. Tepper, P. Musé, A. Almansa, and M. Mejail, "Boruvka meets nearest neighbors," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2013, vol. 8259 LNCS, no. PART 2, pp. 560–567.

[17] L. Dagum, E. Rameshm, and R. Menon, "OpenMP: An Industry- Standard API for Shared- Memory Programming," Comput. Sci. {\&} Eng. IEEE, vol. 5, no. 1, pp. 46–55, 1998.

[18] S. Zhang, Z. Xia, R. Yuan, and X. Jiang, "Parallel computation of a dam-break flow model using OpenMP on a multi-core computer," J. Hydrol., vol. 512, pp. 126–133, 2014.

[19] W. Jeun, Y. Kee, and S. Ha, "Improving performance of OpenMP for SMP clusters through overlapped page migrations," in International Workshop on OpenMP (IWOMP'06, 2006.

[20] B. Chapman, G. Jost, and R. Van Der Pas, Using OpenMP: Portable Shared Memory Parallel Programming, vol. 10. 2008.

[21] Vikas, N. Giacaman, and O. Sinnen, "Multiprocessing with GUI-awareness using OpenMP-like directives in Java," Parallel Comput., vol. 40, no. 2, pp. 69–89, 2014.

[22] R. Van Der Pas, "An Introduction Into OpenMP," Eur. J. Surg., vol. 160, no. 3, pp. 145–151, 2005.

[23] E. Ayguad, N. Copty, A. Duran, J. Hoeflinger, Y. Lin, F. Massaioli, X. Teruel, P. Unnikrishnan, and G. Zhang, "The design of OpenMP tasks," IEEE Trans. Parallel Distrib. Syst., vol. 20, no. 3, pp. 404–418, 2009.

[24] G. Bradski, "The OpenCV Library," Dr Dobbs J. Softw. Tools, vol. 25, pp. 120–125, 2000.

[25] H. Jegou, M. Douze, and C. Schmid, "Hamming embedding and weak geometric consistency for large scale image search," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2008, vol. 5302 LNCS, no. PART 1, pp. 304–317.