Vectorization of a statistical segmentation
Mohammed Elhassani, Delphine Rivasseau, Marc Duranton, Stéphanie Jehan-Besson, David Tschumperlé, Luc Brun, Marinette Revenu

To cite this version:

HAL Id: hal-00083561
https://hal.archives-ouvertes.fr/hal-00083561
Submitted on 23 Sep 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Abstract

We propose an efficient vectorial implementation of a region merging segmentation algorithm. In this algorithm the merging order is based on edge value, and the merging predicate exploits recent statistical investigations. A notable acceleration is obtained by exploiting two forms of parallelism, firstly the Data Level Parallelism by processing edges of the same weight in parallel, secondly the Instruction Level Parallelism. Moreover, the classical UNION-FIND data structure is improved by using local registers to reduce the access time of FIND operations. Finally the implementation could be easily tuned to extract textures (object analysis) or all edges (image enhancement).

INTRODUCTION

Researchers have been working on image segmentation for more than 30 years. Image segmentation is an ill-defined problem, so the optimal solution could not exist, and until now no standards were defined for this field. Nevertheless, many applications could benefit widely from a good segmentation algorithm, for example object oriented compression, pattern recognition, 2D/3D conversion and many others. The image segmentation algorithms could be classified into two categories, namely contour-based and region-based methods. In the first category we find out the significant object boundaries and extract connected components [1]. The main difficulty in this category is to find boundaries closed over objects especially in noisy images. Moreover this approach doesn’t benefit from statistical properties of the image. Because of these limitations, the second category, i.e. region-based, is more often used. In these methods, we merge neighbours regions that verify a certain similarity criterion. Two important points define completely a region-based algorithm; the first one is the similarity criterion used to indicate whether two regions should merge or not, the second one is the order of merging and we present the similarity criterion. As far as notations are concerned, let consider a tree structure and propose two merging order, “mergesquare”, which is claimed as a parallel algorithm, and “scanline”, which is sequential. The main drawback of these orders of merging is that they don’t depend on the image content, which influence the segmentation result. The region adjacency graph approach (RAG) avoids this drawback. In this approach, we can achieve the best local merge, i.e. every region will merge with the most similar of its neighbours. In [7], a Valued region adjacency graph is computed and decomposed in a set of partial complete graph. The RAGs are also used in pyramidal structure [9, 10, 11]. But RAGs approaches still don’t exploit global information of the image. In the implementation point of view, segmentation algorithms are very computing-intensive. Many works proposed parallel algorithms of segmentation to solve the implementation issue. The irregular pyramids were particularly designed to fit a massively parallel architecture. We can also cite [12] in the scope of parallelizing segmentation algorithms. But all these works don’t conciliate the exploitation of global information with the parallelization issue.

In this paper we propose an implementation tending toward this conciliation. We use the algorithm proposed in [13] which combine an order of merging that depends on the content of the image with an adaptive threshold for fusion. We propose an original implementation where the main parts of the algorithm are simplified or vectorized. In the following sections, firstly the algorithm is described, then we propose some implementation solutions where the main steps of the algorithm are vectorized or simplified, and finally we propose a method to tune the algorithm in order to extract textured regions or to extract edges.

THE SEGMENTATION ALGORITHM

In [13], Nock et al proposed a region-based merging. In this algorithm, they combine a specific order of merge with an original similarity criterion. As far as notations are concerned, let consider an image \( I \). The notations \( h \) and \( w \) denotes respectively the horizontal and vertical size of the image, \( |I| = h \times w \) is the total size of the image, \( a(p) \) is the pixel colour level at position \( p \) and \( g \) denotes the maximum colour level. In the two following sections we explain the order of merging and we present the similarity criterion.

ORDER OF MERGING

The order of merging is built based on the edges values as in [13, 3]. The idea behind this order of merging is to merge first what is similar before merging what is different.

In our algorithm, an edge corresponds to a couple of pixels \((p, p')\) in 4-connectivity. The edge values \( v \) correspond to the maximum of the three differences over the three colour components \((r, g, b)\): 

\[ v(p, p') = \max_{a \in \{r, g, b\}} \left( |a(p) - a(p')| \right). \]

The edges are then sorted in an increasing order of their values and corresponding pixels are treated in this order for fusion.

THE CRITERION OF MERGING

We use the criterion of merging proposed in [13]. Let’s explain briefly how this criterion works. Given two neighbours regions \( s_1 \) and \( s_2 \), the average of the three colour components within these regions are denoted by \( \mu_{a_{1}}, \mu_{a_{2}}, \) with \( a \in \{r, g, b\} \). The region cardinal of \( s_1 \) is denoted \( |s_1| \). The criterion for merging the two regions is the
following:

\[
Pr(s_1,s_2) = \begin{cases} 
true & \Delta H(s_1,s_2) \leq g * f(s_1) + f(s_2) \\
false & \text{otherwise}
\end{cases}
\] (2)

\[
\Delta H(s_1,s_2) = \max_{a \in [p_1,p_2]} (\mu_a - \mu_{a_2}).
\]

The adaptive threshold \( f(s) \) takes into account the region size \(|s|\) as follows:

\[
f(s) = \min(g,|s|) * \frac{\ln(|s|+1) + \ln(\gamma)}{2 * Q * |s|}.
\]

\( \gamma = 6 * |H|^2. \)

This threshold is based on a statistical model of the image and obtained using McDiarmid’s inequality, see[13] for more details. \( Q \) is a parameter set by the user that could tune the coarseness of the segmentation.

### IMPLEMENTATION

As shown in Fig.1 the algorithm can be decomposed in three main steps. The first step performs histogramming where the histogram of edges values is computed. This histogram is then used to order edges. In the third step we do the merging following this order of edges. The parallelism in the three steps is not obvious. To solve this dependency, we use the same idea detailed in equation (2). Indeed the three operations are irregular both in data access order and in computations. In this paper we focus on the vectorization of computation. In this vectorization we process a vector of \( n \) data \( D = [d_1,d_2,...,d_n] \) in parallel way. In the following we detail the vectorization of the main steps of the algorithm, i.e. histogramming, sorting of edges and merging.

### HISTOGRAMMING VECTORIZATION

Let us consider that \( H \) denotes the histogram of edge values that is computed in this step. To compute \( H \), firstly we compute edges values \( v \) as detailed in equation (1), secondly we compute the distribution \( H \) of these values.

There is no data dependency in the computation of edges values, so we can achieve this operation in vectorial way over a vector of edges \( E = [(p_1,p_2),...,(p_{2n-1},p_{2n})] \) which result on a vector of values \( V = [v_1,...,v_{2n}] \). However, computing the distribution \( H \) of the edges values in vectorial way, is not straightforward. Indeed two edges values could be equal, and incrementing the histogram’s bin corresponding to this value in parallel way will give incorrect result. To solve this data dependency, we propose the following method:

We consider an array \( T \) of \( g \) cells, each cell is \( n \) bits width. Each \( v_i \) in \( V \) set the \( i^{th} \) bit of the \( v_i^{th} \) cell of \( T \). Then we add the bits of each cell of \( T \) in one instruction. The result in \( T \) is used to update the histogram \( H \). The algorithm is described in details in Algorithm.1.

From the hardware point of view, this vectorization requires binary adders with \( n \) input, which is very simple.

Let us explain how the hardware \( H \) is used for the sorting step. We consider an array \( M \) of size \( w * (h-1) + (h-1) * w \) which is the number of edges in the 4-connectivity in the whole image. This array \( M \) will be used to store the order of edges. We compute the accumulated histogram \( H_0 \) as detailed in equation (3). This \( H_0 \) is used to partition \( M \) in \( g+1 \) parts, the \( i^{th} \) part is limited between \( H_0[i] \) and \( H_0[i+1] \) addresses. In this \( i^{th} \) part of \( M \) we will store edges with values equal to \( i \).

### SORTING VECTORIZATION

In this step, we want to assign to a vector \( E \) of edges a vector of addresses \( A \) which will be stored in \( M \). There is a data dependency in this step: if two edges have the same value, the assignment of two different addresses to these two edges in parallel way is not obvious. To solve this dependency, we use the same idea detailed in the previous section. Each edge \( E[i] \) with value equal to \( v_i \) set the \( i^{th} \) bit of the \( v_i^{th} \) cell of \( T \). Then we assign to \( E[i] \) an address \( A[i] \) as detailed in Algorithm.2.

### MERGING VECTORIZATION

The algorithm of merging is described in Algorithm.3. This algorithm uses the UNION-FIND data structure. For an edge that corresponds to a couple of pixels \((p_1,p_2)\), we use the “FIND” operation to find the couple of segments \((s_1,s_2)\) containing these two pixels, then the predicate is evaluated for \((s_1,s_2)\) as described in equation(2). If the predicate is true, we make the “UNION” of \( s_1 \) and \( s_2 \). After the “UNION” operation, we compute the new segment properties \((|s|,\mu_x,\mu_y,\mu_o)\) and update the main memory with these information. In this paper we focus on the vectorization of the predicate evaluation and the “UNION” operation for a vector \( S \) of couples \((s_1,s_2)\). The FIND operation still be hard to parallelize.

Firstly we propose a simplification for the thresholds computation detailed in equation (2). We used a linearization by di-
The first part of the algorithm describes how to vectorize the merging step in the sorting process. It involves iterating over the elements of a vector $S$ and updating a matrix $A[i]$ for each element. The update is done in parallel, allowing for efficient processing of the vector.

Algorithm 2 Vectorization of sorting

```plaintext
for $i \in [0 : n - 1]$ do
    $T[i] = 0$
end for

for $i \in [0 : n - 1]$ do
    $T[v[i][i]] = 1$
end for

for $i \in [0 : n - 1]$ do
    $A[i] = H_0[v[i]] + H[v[i]] + \sum_{j=0}^{l-1} T[v[i][j]]$
end for

for $i \in [0 : n - 1]$ do
    $H[v[i]] = H[v[i]] + \sum_{j=0}^{l-1} T[v[i][j]]$
end for
```

This algorithm leverages vectorization to optimize the merging process, reducing the number of operations required for sorting. The vectorization allows for parallel processing of multiple elements simultaneously, leading to improved performance.

Algorithm 3 Vectorization of sorting

```plaintext
for all the edges in the sorted list do
    p1 and p2 are the pixels connected by the edge
    $s_1 = FIND(p1)$
    $s_2 = FIND(p2)$
    if ($Pr(s1, s2) = True$) then
        $UNION(s1, s2)$
    end if
end for
```

This algorithm extends the vectorization to include edge processing, enabling parallel computation of edge-related operations. The vectorization of edge operations can further enhance performance gains in image processing tasks.

The locality of data and the parallelism of operations are crucial factors in achieving good performance in these algorithms. Vectorization techniques can significantly reduce the computational complexity, making the algorithms more efficient for large datasets.

In conclusion, the vectorization of operations allows for the parallel processing of data, which is essential for improving the efficiency of algorithms like sorting and image processing. The use of vectorization and parallelism can lead to substantial speedups, making these algorithms more effective and scalable for modern computing environments.
TUNING THE SEGMENTATION

The image segmentation requirements are different depending on the application. In Image enhancement, the main properties to find are edges in order to process pixels belonging to homogeneous regions in the same manner, while in many image analysis applications like pattern recognition, texture extraction is fundamental. We propose a very simple method to switch the segmentation from a texture-oriented to an edge-oriented segmentation. When using the predicate of equation (2), we find out textures. But if we replace this predicate by the one described in equation (4), we will find out all the edges higher than a fixed threshold.

\[ P(s_1, s_2) = \begin{cases} 
\text{true} & \text{if } \Delta v(p_1, p_2) \leq tr \\
\text{false} & \text{otherwise} 
\end{cases} \]

(4)

\[ \Delta v(p_1, p_2) = \max_{a \in [r, g, b]} (a_1 - a_2). \]

Where \((p_1, p_2)\) is the edge being processed, \(s_1, s_2\) are segments containing \(p_1\) and \(p_2\), \(tr\) is a threshold fixed experimentally to 10 for \(g = 255\). In Fig.4 we show the result of segmentation of one image for the two predicates. In Fig.4(a) we show the result of a textured-oriented segmentation with the first predicate of one textured image. Notice that the textures are well segmented. In Fig.4(b) we show the result of an edge-oriented segmentation by using the second predicate. We can see all the edges in white colour.

CONCLUSION

Actually we are investigating to build a memory system where data is accessed by content instead of address. With such system we will implement the “FIND” operation efficiently. The solutions proposed in this paper were tested in c language and we are looking for a real hardware implementation.

References


**Author Biography**

Mohammed El Hassani was born in Fkih Ben Salah, Morocco, in 1979. He received the engineering degree in Electronics from the ENSERG school, Grenoble, France, in 2002 and the Graduate degree in Microelectronics from the INPG in 2003. He is a Ph.D. candidate at PHILIPS Caen in collaboration with Caen University. His research interests are in video processing and parallel architecture.