Paper: Vectorization of a statistical segmentation

M. El Hassani, D. Rivasseau; Philips Semiconducteurs, Caen, France.
M. Duranton; Philips Research Eindhoven, Netherlands.
S. Jehan-Besson, D. Tschumperle, L. Brun, M. Revenu; Laboratory GREYC, Caen University, France.

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, 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 be merged or not, the second one is the order in which the similarity test should be done. There is an important gap about the way these two points interact. Many similarity criterions have been used in the segmentation literature, In [2] the most used criterion are reviewed, In [3, 4, 5] some robust criterion are proposed. These similarity criterions are combined with a data structure that establishes an order in similarity test. In [6], they use 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. In the region adjacency graph approach (RAG), we can achieve the best local merge, i.e. every region will merge with the most similar of its neighbours. In [7, 8], 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 do not 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 denote respectively the horizontal and vertical size of the image, $|I| = h * w$ is the total size of the image, $a(p)$ is the pixel colour level at position p. 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, we will refer to an edge by its position e. The edge value v corresponds to the maximum of the three differences over the three colour components $\{R,G,B\}$:

$$v(p,p') = \max_{a\in\{R,G,B\}} \left(|a(p) - a(p')|\right). \quad (1)$$

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 segments s_1 and s_2, the average of the three colour components within these segments are denoted by μ_{a_1}, μ_{a_2} with $a \in \{R,G,B\}$. The segment cardinal of s_i is denoted $|s_i|$. The criterion for merging the two segments is the following:

$$Pr(s_1,s_2) = \begin{cases} true & if \;\; \Delta\mu(s_1,s_2) \le g * \sqrt{f(s_1)+f(s_2)} \\ false & otherwise \end{cases} \quad (2)$$

$$\Delta\mu(s_1,s_2) = \max_{a\in[R,G,B]} (\mu_{a_1} - \mu_{a_2}).$$

The adaptive threshold $f(s_i)$ takes into account the segment size $|s_i|$ as follows:

$$f(s_i) = min(g,|s_i|) * \frac{\ln(|s_i|+1)+\ln(\gamma)}{2*Q*|s_i|}. \quad \gamma = 6*|I|^2.$$

Where the parameter g corresponds to the maximum colour level, and Q is a parameter set by the user that could tune the coarseness of the segmentation. This threshold is based on a statistical model of the image and obtained using McDiarmid's inequality, see[13] for more details.

IMPLEMENTATION

As shown in Fig.1 the algorithm can be decomposed in three main steps. The first step corresponds to the computation of the histogram of edges. This histogram is then used to order edges. In the third step, the merging is performed following this order of edges. The parallelism in the three steps is not obvious to extract. Indeed the three operations are irregular both in data access from the memory and in computations. In this paper we focus on the vectorization of computation. In this vectorization we process in parallel a vector of n data $D = [d_1, d_2 ... d_n]$. 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 a vectorial way over a vector of edges $E = [e_1, e_2 ... e_n]$, which results on a vector of values $V = [v_1 ... v_n]$. However, computing the distribution H of the edges values in a vectorial way, is not straightforward. Indeed two edges values could be equal, and incrementing the histogram's bin corresponding to this value in a parallel way will give incorrect result. To solve this data dependency, we consider an array T of $g+1$ cells, each cell is n bits width. Each v_i in V sets 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 a hardware point of view, this vectorization requires binary adders with n inputs, which is very simple.

Let us explain how the histogram H is used for the sorting step. We consider an array M_v of size $h*(w-1)+(h-1)*w$, which is the number of edges in the 4-connectivity in the whole image. This array M_v will be used to store the order of edges. We compute the accumulated histogram H_a as detailed in the following equation:

$$\begin{aligned} H_a[0] &= 0; \\ H_a[i] &= H[i-1]+H_a[i-1]; \end{aligned} \quad (3)$$

Algorithm 1 Vectorization of histogramming

for $i \in [0:g]$ **do**
  for $j \in [0:n-1]$ **do**
    $T[i][j] = 0$
  end for
end for
for $i \in [0:n-1]$ **do**
  $T[v_i][i] = 1$
end for
for $i \in [0:n-1]$ **do**
  $H[v_i] = H[v_i] + \sum_{j=0}^{j=n-1} T[v_i][j]$
end for

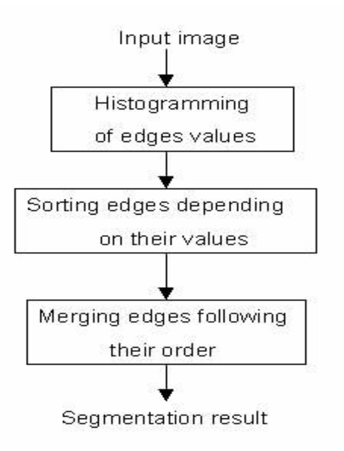

Figure 1. *General diagram of image segmentation.*

The accumulated histogram H_a is used to partition M_v in $g+1$ parts, the i^{th} part is limited between $H_a[i]$ and $H_a[i+1]$ addresses. In this i^{th} part of M_v 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 where they will be stored in M_v. 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 as detailed in the previous section. We reinitialise the histogram H to zero. Each edge $E[i]$ with value equal to v_i sets the i^{th} bit of the v_i^{th} cell in T. Then we assign to $E[i]$ an address $A[i]$ based on H_a and the re-computation of H 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. Let's assume that s_1 becomes the parent After the "UNION" operation, then we update the properties of s_1 ($|s_1|$,sum_{R_1}, sum_{G_1}, sum_{B_1}), where the notation sum_{a_1}, denotes the sum of the a component for all the pixels belonging to the segment s_1. These information are updated in the main memory after the processing of each edge. In

Algorithm 2 Vectorization of sorting

for $i \in [0:g]$ **do**
  for $j \in [0:n-1]$ **do**
    $T[i][j] = 0$
  end for
end for
for $i \in [0:n-1]$ **do**
  $T[v_i][i] = 1$
end for
for $i \in [0:n-1]$ **do**
  $A[i] = H_a[v_i] + H[v_i] + \sum_{j=i}^{j=n-1} T[v_i][j]$
end for
for $i \in [0:n-1]$ **do**
  $M_v[A[i]] = E[i]$
end for
for $i \in [0:n-1]$ **do**
  $H[v_i] = H[v_i] + \sum_{j=0}^{j=n-1} T[v_i][j]$
end for

Algorithm 3 Merging process

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

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 is still hard to parallelize.

Firstly we propose a simplification for the thresholds computation detailed in equation (2). We use a piecewise linear approximation of the function f. This linearization, which is obtained by dichotomy, provides a Look Up Table. To compute a threshold value f corresponding to one segment cardinal $|s_i|$, we read the coefficients α and β from the LUT, and the computing is $f(s_i) = \alpha * |s_i| + \beta$. This trick simplifies the computation a lot without a visible loss of quality in the segmentation result.

Let us explain how the vectorization of the merging step is achieved. We consider a vector of couples of segments S, which is the result of the "FIND" operation applied on a vector of edges E. We load the vector S and the data corresponding to each segment s_i ($|s_i|$, sum_{R_i}, sum_{G_i}, sum_{B_i}) in local registers. Processing the vector S in parallel is not straightforward because one segment label s_i could be equal to another s_j, and the result of merging will be incorrect as described in the example Fig.3. So the level of parallelism depends highly on the image content.

The vectorization of the merging of S is done in the following way. Firstly the elements of the vector S are classified into two parts, the first one contains independent couples (s_i, s_{i+1}) where one label s_i figures once only, the second part contains dependent couples where one label s_i figures in another couple in S. Firstly we process the first part of S in parallel way. Secondly we process in sequential way the second part. We then investigated the optimal width n of the vector S that gives the best data level parallelism (DLP) exploitation. In this investigation, we compute the ratio between the number of operations in the vectorized merging, over the number of operations in the sequential merging described in Algorithm 3. In Fig.2 we show this ratio for different vector widths and for many real sequences. The acceleration is maximal for a vector width around 25.

	n= 20	n= 25	n= 50	n=100	n= 200	n= 300	n= 400	n= 500	n= 600
Bicycle	1.310	1.312	1.308	1.296	1.281	1.273	1.269	1.266	1.264
Boat	1.504	1.510	1.501	1.475	1.454	1.446	1.442	1.439	1.437
Soccer	1.438	1.446	1.443	1.423	1.405	1.398	1.393	1.391	1.389
Sport	1.297	1.305	1.315	1.304	1.286	1.279	1.275	1.273	1.271

Figure 2. Ratio between the number of operations of the sequential merging and the merging when exploiting data level parallelism.

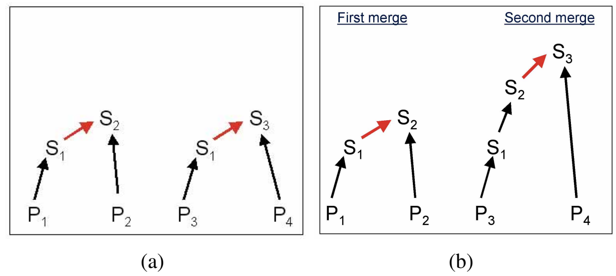

Figure 3. Merging two couple of segments (s_1, s_2) and (s_1, s_3): a. Parallel merge, after merging, s_1 belongs to s_2 and s_3, which are two different segments, so the result is incorrect. b. Sequential merge : after the first merge s_1 belongs to s_2, after the second merge both s_1 and s_2 belong to s_3, the result is correct

In addition to the DLP exploited by processing the first part of S in parallel, the processing of the second part of S benefits from the locality of data. Indeed, if two couples (s_i, s_{i+1}), (s_j, s_{j+1}) share one label, they are processed sequentially. The latest one will use the result of merge of the first one, which is still be available in local registers, instead of getting it from the main memory as in the simple sequential merging.

In the other hand when evaluating the predicate and updating segment's properties, some of the operations are independent and could be parallelized. Let us consider that s_1 and s_2 are merged, and s_2 becomes the representative of the two segments. The operations used for the evaluation of the predicate and for updating are :

-Predicate evaluation:
$f(s_1) = \alpha_1 * |s_1| + \beta_1; f(s_2) = \alpha_2 * |s_2| + \beta_2;$
$\mu_{a_1} = sum_{a_1}/|s_1|; \mu_{a_2} = sum_{a_2}/|s_2|;$
$\Delta\mu(s_1, s_2) = \max_{a \in [R,G,B]}(\mu_{a_1} - \mu_{a_2});$
$\Delta\mu^2(s_1, s_2) = \Delta\mu(s_1, s_2) * \Delta\mu(s_1, s_2); tr = f(s_1) + f(s_2);$
$Comparison(\Delta\mu^2(s_1, s_2), tr);$

-Updating of segment properties:
$|s_2| = |s_1| + |s_2|; sum_{a_2} = sum_{a_1} + sum_{a_2};$

There are 18 operations in the processing of the couple (s_1, s_2). If we have enough resources (4 ALUs, 4 MACs, 6 Divider), we can use the instruction level parallelism (ILP), and do this processing in six steps.

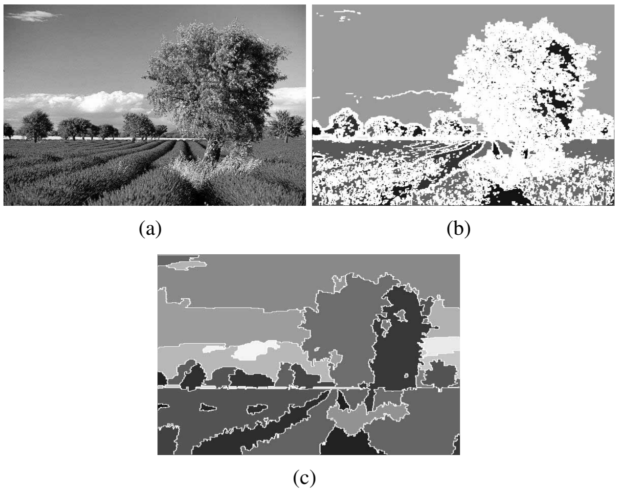

Figure 4. Tuning segmentation: a. The original image. b. An edge-oriented segmentation. c. A texture-oriented segmentation

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 segments 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.

$$P(s_1,s_2) = \begin{cases} true & if \;\; \Delta v(p_1,p_2) \leq tr \\ false & otherwise \end{cases} \tag{4}$$

$$\Delta v(p_1,p_2) = \max_{a \in [R,G,B]} (a_1 - a_2).$$

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 a system, we will implement the ”FIND” operation efficiently. The solutions proposed in this paper are tested in C language. Our research is now directed toward a study of a real hardware implementation of these ideas.

References

[1] G. Iannizzotto and L. Vita, “Fast and accurate edge-based segmentation with no contour smoothing in 2-d real images,” *IEEE Transactions on Image Processing*, vol. 9, Issue 7, pp. 1232 – 1237, 2000.

[2] S.Bres, J.Jolion, and F.Lebourgeois, ,” in *Traitement et analyse des images numérique*. Herms - Lavoisier, 2003, pp. 251–252.

[3] P.F.Felzenszwalb and D.P.Huttenlocher, “Efficient graph-based image segmentation,” *International Journal of Computer Vision*, vol. 59, Issue 2, pp. 167–181, 2004.

[4] S.Lallich, F.Muhlenbach, and JM.Jolion, “A test to control a region growing process within hierarchical graph, 36,” *Pattern Recognition*, pp. 2201–2211, 2003.

[5] C.Fiorio and R.Nock, “Image segmentation using a generic, fast and non-parametric approach tools with artificial intelligence,” *Tenth IEEE International Conference*, pp. 450–458, 1998.

[6] C.Fiorio and J.Gustedt, “Two linear time union-find strategies for image processing,” *Theoretical Computer Science*, vol. 154, pp. 165–181, 1996.

[7] Jianibo Shi and J. Malik, “Normalized cuts and image segmentation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, Issue 8, pp. 888 – 905, 2000.

[8] E. Sharon, A. Brandt, and R. Basri, “Fast multiscale image segmentation,” *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 1, pp. 70 – 77, 2000.

[9] J.M.Jolion, “Stochastic pyramid revisited,” *Pattern Recognition Letters, 24*, vol. 24, pp. 1035–1042, 2003.

[10] Y.Haximusa, A.Ion, W.G.Kropatsch, and L.Brun, “Hierarchical image partitioning using combinatorial maps,” *Joint Hungarian-Austrian Conference on Image Processing and Pattern Recognition*.

[11] W.G.Kropatsch and S.Ben Yacoub, “A revision of pyramid segmentation,” *Proceedings of the 13th International Conference on Pattern Recognition*, vol. 2, pp. 477–481, 1996.

[12] B.Gallile, M.Renaudin, P.-Y.Coulon, and F.Mamalet, “Algorithme-architecture parallèle asynchrone pour la segmentation d’image par ligne de partage des eaux,” *CORESA*, 2001.

[13] R. Nock and F. Nielsen, “Statistical region merging,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, 2004.

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.