Multiresolution Watershed Segmentation on a Beowulf Network

Syarraieni Ishar[1] and Michel Bister[2]

[1]Faculty of IT, Multimedia University, 63100 Cyberjaya, Malaysia,
syarraieni.ishar@mmu.edu.my
[2]Faculty of Engineering, Multimedia University, 63100 Cyberjaya, Malaysia,
mrbister@ieee.org

Abstract. Among the many existing multiresolution algorithms, the scale-space approach offers the benefits of strong mathematical and biological foundation and excellent results, but the serious drawback of a heavy computational load. Parallel implementation of this category of algorithms has never been attempted. This article presents a first experiment, using the multiresolution watershed segmentation as algorithm, and an 8-node Beowulf network as hardware platform. First, the classical approach is followed whereby the image is divided in several regions that are separately allocated to different nodes. Each node performs all the calculations for his region, at any level of resolution. Next, a truly multiresolution approach is followed, allocating the workload to the processors per resolution levels. Each node is allocated a number of resolution levels in the scale space, and performs the calculations over the whole image for the particular resolution levels assigned to it. The implementation in the latter approach is clearly much more straightforward, and its performance is also clearly superior. Although the experiments using the region-wise assignment were only done by splitting up the image in rows, and not in columns or in quadrants, the difference in the results is so dramatic that the conclusions can easily be generalized, pointing to the fact that scale space algorithms should be paralellised per resolution level and not per image region.

1 Introduction

1.1 Scale Space

Over the past few years, multiresolution algorithms have become a dominant approach in digital image processing, and this for two good reasons: their excellent results, and their biological motivation. It has indeed been shown that the human visual system is largely a multiresolution system. The scale space approach is an alternative to other several approaches that have been suggested: Laplacian pyramids, filterbanks, wavelets, etc [1].

The scale space approach first develops its theory in the continuous domain, and adds a scale axis to the spatial axes of the image. For practical applications, the scale axis has to be sampled, just as the spatial axis is sampled. The whole development of the theory is aimed at studying the evolution of the image features along scale. This is termed the "deep structure".

H. Kosch, L. Böszörményi, H. Hellwagner (Eds.): Euro-Par 2003, LNCS 2790, pp. 624–631, 2003.

Since an *N*-dimensional image is extended over an additional dimension (scale), it is obvious that this approach is both memory- and computational intensive - hence the greater popularity of sub-sampled alternatives like wavelets. However, the scale space has much stronger mathematical motivations, as it can theoretically be deduced from basic assumptions in many different ways, including causality, dimension analysis, entropy maximization, etc. Also, the correlation between the findings from the scale space approach and the human front end vision (the uncommitted part of the human visual system) is striking. Finally, results are impressive. Most of the "classical" digital image processing techniques (e.g. deblurring, texture analysis, gradient analysis, optical flow, stereo analysis) have all been implemented using scale space techniques [2, 4, 5, 8].

Taking into account the heavy demands of any scale space algorithm, it is obvious that the scale space community would benefit greatly from parallel implementations. Hence, it is surprising that no parallel implementation of the scale space approach has been attempted to date. This paper is a first attempt at filling this void. Taking into account the vast amount of different scale space algorithms, it was necessary to limit the selection.

1.2 Watershed Segmentation

The basic principle of watershed segmentation is to let the pixels in the image link to one another, whereby each pixel links to its neighbor with the least gradient magnitude - hence away from the strongest gradient. This linking scheme defines a classification, whereby all the pixels linking mutually define one class, or segment. In practice, the links are propagated from one pixel to another: if pixel A links to pixel B, and pixel B links to pixel C, then pixel A is linked directly to pixel C. This is iterated until there are no more changes. Pixels who link to themselves (e.g. pixel X links to pixel Y, and pixel Y links to pixel X, so after one iteration pixel X links to itself) are called roots. Roots are numbered, and all pixels are labeled with the number of the root to which they link.

This algorithm is illustrated in Figure 1. It is called watershed segmentation for the following reason. If we consider the gradient image as a landscape, with the magnitude of the gradient as a measure of the elevation of the point, then the algorithm simulates the flow of rain and the splitting up of the landscape in watersheds. Several implementations exist, not all following the simple and intuitive procedure outlined here.

The main problem with watershed segmentation is its inherent over-segmentation: the smallest local minimum in the gradient produces a new root, hence a new segment. Many methods have been proposed in the literature to deal with this problem, including the low-pass filtering of the image to reduce the noise and hence the number of local minima in the gradient image. The question is then: how much filtering should be applied? The scale space approach gives an elegant answer to this question by relating the results obtained at different levels of resolution.

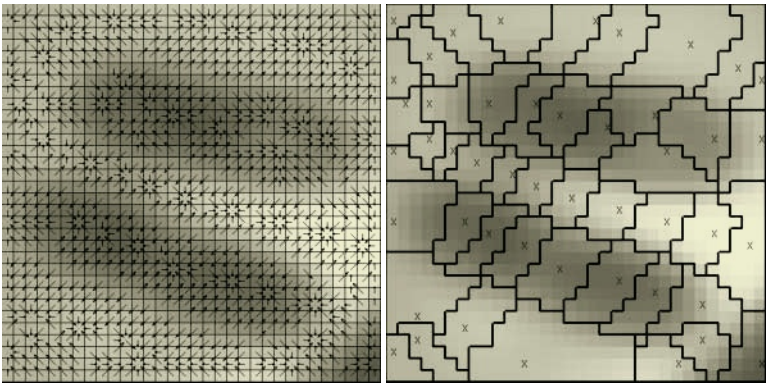

Fig. 1. Watershed algorithm of a bacteria image. Left: linking scheme (pixels link toward smallest gradient). Right: roots are marked by the letter 'x', and dark lines show the border of the segments.

1.3 Scale Space Watershed

Let us imagine an image blurred with two different levels of blurring, σ_1 and σ_2. Each image is segmented independently using the watershed algorithm. The segmentation using σ_1 is blurred just enough to have sufficiently smooth borders but too many segments, while the segmentation using σ_2 is too blurred for the borders (the borders have shifted) but just enough to avoid over-segmentation. The purpose would be to have the segments as defined in the σ_2 image but with the borders as determined in the σ_1 image. The problem is to trace the borders of the segments from the higher level in the scale space (associated with σ_2) to a lower level (associated with σ_1). As tracing the borders of a segment corresponds to tracing the segment, the maximum overlap algorithm, as illustrated in Figure 2, is used as the solution.

The segments at level σ_1 are named S_i, $0 \le i < N_{s1}$, and the segments at level σ_2 are named T_j, $0 \le j < N_{s2}$. Then, the amount of overlap $V(S_i, T_j)$ is equal to the number of pixels that belong both to segment S_i at level σ_1 and to segment T_j at level σ_2. Segment S_i is linked to segment T_j if and only if $V(S_i, T_j) > V(S_i, T_k)\ \forall\ k \neq j$. Finally, all segments at level σ_1 that link to the same segment at level σ_2 are merged, resulting in a number of segments equal to the N_{s2} but with segment borders defined at level σ_1.

This procedure can be propagated throughout the scale space, as in a loop, so as to have segments defined at any level of scale with borders defined at another level of scale. This is the algorithm that we want to implement here. Hence, several steps have to be considered: blurring, gradient calculation, watershed segmentation, and merging. Actually, we want to leave the merging as a parameter to the user, so we want to generate the structure of segment linking from one level to another.

	1	2	3	4	5
1	4	2	-	-	-
2	-	3	6	1	-
3	-	-	3	1	-
4	-	7	-	-	-
5	-	-	1	4	-
6	5	2	-	-	-
7	-	8	-	-	-
8	-	-	-	-	13
9	-	-	-	4	-

Fig. 2. Scale space merging algorithm. From left to right: gray scale input image; result of segmentation on level σ_1 – segment borders accurate, but over-segmentation; result of segmentation on level σ_2 – reduced over-segmentation, but segment borders inaccurate due to blurring; overlap table between the two segmentations; resulting of assigning to each segment at level σ_1 the value of the segment at level σ_2 with which it has the highest overlap.

1.4 Beowulf Networks

The Beowulf system provides a low-cost high-performance parallel computing environment. This system is ideal for loosely coupled parallel processing which involves a set of independent processors, each with its local memory, typically with their own copy of the OS, working in parallel to solve a problem. The system consists of one server node interconnected with clients via a high-speed network, allowing each processor to work on its own data [9].

Beowulf cluster is used to run the SPMD (Single Program Multiple Data) algorithm using either of the two following popular libraries: Message Passing Interface (MPI) or Parallel Virtual Machine (PVM) [10]. MPI is a popular message passing interface used in numerous homogeneous hardware environments ranging from distributed memory parallel machines to networks of workstations [3, 7, 10].

2 Implementation

In our parallel implementation of the scale space watershed algorithm, we use a Beowulf cluster consisting of 4 Dual Processors Intel Pentium III 800MHz workstations (1 master and 7 slaves) with 512Mb, 20.4Gb, running RedHat Linux 7.0 operating systems. These workstations are interconnected via a Myrinet switch.

2.1 Region-Wise Implementation

There are several parallel processing approaches for image data such as segment processing, frame processing, column-wise processing, row-wise processing, etc. For solving the parallel watershed problem, the row-wise approach was first implemented. Distribution of the working area over the processors or slaves is based on the number of rows in the input image. Each slave works on a sub-image for the watershed segmentation for all scale levels.

Determining of links only requires the sharing of one row of pixels along the border of the sub-region. Hence this poses no problems. Likewise, the numbering of

the roots can be done easily, with just a little precaution to make sure that each sub-region gets unique root numbers.

The real problem comes from the link following part of the algorithm. Several possible situations are illustrated in Figure 3, where an 8x6 image was divided among 3 slaves. A first linking list is without problem: link(A) = B, link(B) = C, so the instruction link(A) = link(link(A)) results in link(A) = C, after which iterations stop because link(link(A)) == link(A). All of this can be done within the region assigned to slave 1. The second case (pixels D-E-F-G) illustrates the need for message passing between processors: but F is outside the region assigned to slave 1, so slave 1 needs information from slave 2 to continue following the list and put link(D) to G. One could consider letting each slave first iterate until no further changes *within its assigned region*, and then exchange all the link information from the border pixels. In this case, slave 1 would already know that link(D) = F and slave 2 know that link(F) = G, so passing this information to slave 1 would allow this one to put link(D) equal to G.

Further problems are illustrated in the third case: repeated crossing of the border line (from H to Q…) or linking between regions that are not even adjacent (from R to W…), which illustrate that 1) just linking within each region and then sharing border information is not enough; 2) sharing information once between adjacent regions is not enough - repeated sharing OR sharing between all regions is necessary.

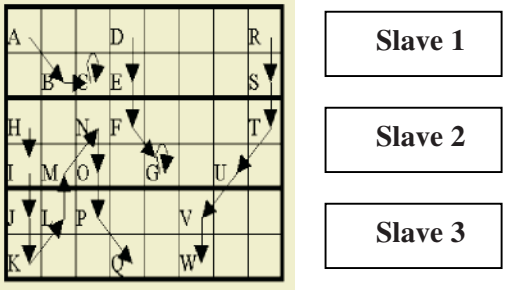

Fig. 3. Possible situation encountered when following linked lists of pixels between regions assigned to different slaves (see text for explanation).

Since the linking algorithm requires updated information of other pixels that are scattered throughout the image, and not only in the neighboring sub-images, many message-passing activities would occur. Every slave would communicate to each other slave to gather other sub-images information. Many synchronization points would be added for this implementation. To simplify the implementation, instead of many individual message passing steps, one single centralised message passing step is implemented: after slaves have completed the watershed segmentation for every level, they will send their result to master. Once gathered, master will display the findings.

2.2 Scale-Wise Implementation

In a second implementation, the task is distributed among the processors based on the scale levels. The master will first calculate the number of scale levels based on the

following formula and distribute the range of scale levels to all slaves as in Figure 4 [4]:

$$\text{Scale level } (\sigma) = \log(\text{Number of extrema in input image})/2*\text{rate}$$

Each slave will then perform a sequential watershed algorithm. Once the watershed segmentation is completed for all levels assigned to them, the slaves send their results to the master. A few synchronization points were added to the code to ensure that the master receives the correct segmentation level. The master will then display the watershed segmentations.

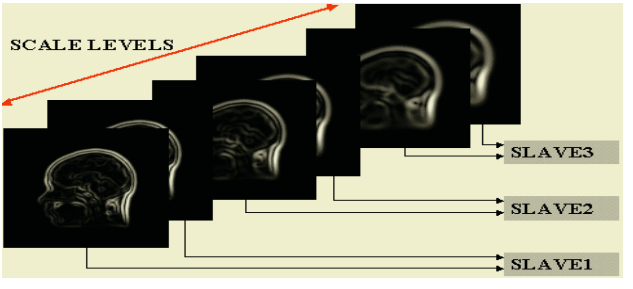

Fig. 4. Parallel implementation of the scale space watershed algorithm per level.

3 Results

The two implementations were applied on many different popular test images in digital image processing, but due to the space constraints, only the results for 8 of them are reported here: 4 with resolution 512x512 (Airplane, Boat, Crowd, Peppers) and 4 with resolution 256x256 (Camera, Couple, House, Orca). Thumbnails of these images are given in Figure 5. For each of these images, the experiments were repeated 4 times for more reliable results.

The average execution time for each of these images and for all three implementations is recorded: serial (for reference), per row, and per level. From this, the average speedup factor was determined using the formula: Speedup S(n) = Execution Time (1 processor) / Execution Time(n processors) = t_s/t_p [10]. All the results are shown in Figure 6.

Fig. 5. Thumbnail of 8 test images used in the experiments.

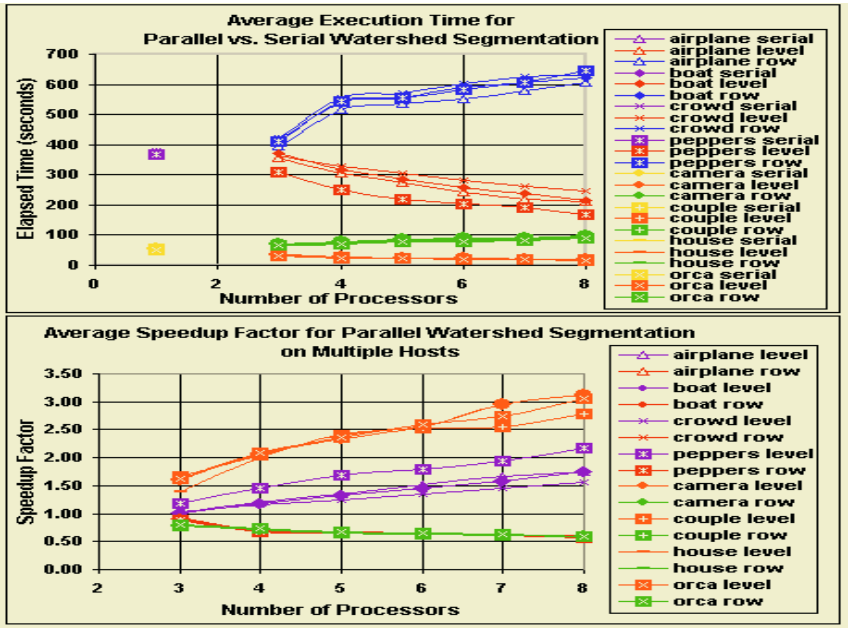

Fig. 6. Results for each test image in function of number of processors: a) average execution time; b) average speedup factor.

4 Discussion

From the results it is clear that the implementation by scale levels is much superior to the implementation by rows. The implementation by rows systematically slows down the process. This is due to the intensive message-passing during the linking phase of the watershed algorithm. As was shown previously, at this stage, each processor needs a lot of interaction with all the other processors. This creates a communication bottleneck, which only increases with increased number of processors.

On the other hand, the implementation by scale levels systematically speeds up the process. This is the logical choice, since the watershed is typically performed independently at each level of the scale space, minimized message passing. As a result, we have a speedup for the implementation by scale levels, but an actual speed-down in the implementation by rows.

The performance is strongly dependent on image size and image structure. For small images, obviously the execution time is shorter, but also the speedup factor for the implementation by level is higher. For the 512x512 images, the results vary more, with the lowest to the highest speedup, (speedup of 1.56 (Crowd) to 2.18 (Peppers) with 8 processors). Regardless the image size, the highest speedup is the Orca (factor 3.06). It is tempting to explain this by the lower complexity of the Orca image (large homogenous regions) and the high complexity of Airplane and Crowd images.

5 Conclusion

The large number of tests performed and the consistency of the results are enough to show that the parallellisation of the scale space watershed algorithm should be done by level and not by row. The reason for the failure of implementation by rows is the intense message-passing required during the linking phase of the watershed algorithm. Any other region-wise implementation (by quadrant, by column, etc) would face the same problem.

On the other hand, the implementation by resolution levels is coarsely grained because the processing is done independently per resolution level. As the number of processors increases, fewer and fewer levels are assigned per processor. The speedup curve levels off as one approaches one level per processor - beyond this limit, it is clear that no additional speedup could be expected any more. Particularly for the smaller images, with a smaller number of levels, the leveling of the speedup curve is clearly visible.

Since many deep structure algorithms work independently on individual levels or on pairs of levels, similar performance would be expected, and the logical choice would definitely be the implementation of scale space algorithms by levels of resolution.

References

1. Bister M, Cornelis J, Rosenfeld A, A critical view on pyramid segmentation algorithms. Pattern Recognition Letters, Vol. 11, pp. 605–617, 1990.
2. Florack, L.M.J. Image Structure. Computational Imaging and Vision Series. Kluwer Academic Publishers, Dordrecht. (1997)
3. Gropp, W., Huss-Lederman, S., Lumsdaine, A., Lusk, E., Nitzberg, B., Saphir, W., Snir, M.: MPI- The Complete Reference: Volume 2, The MPI Extensions. The MIT Press, Cambridge London. (1998)
4. Romeny, B.M. ter Haar (ed): Front-End Vision and Multiscale Image Analysis. Kluwer Academic Publishers, Dordrecht. (2003)
5. Romeny, B.M. ter Haar (ed): Geometry-Driven Diffussion in Computer Vision. Kluwer Academic Publishers, Dordrecht. (1994)
6. Snir, M., Otto, S., Huss-Lederman, S., Walker, D., Dongarra, J.: MPI – The Complete Reference: Volume 1, The Core. 2nd edn. The MIT Press, Cambridge London. (1998)
7. Sporing, J., Nielsen, M., Florack, L., Johansen, P. (eds): Gaussian Scale-Space. Kluwer Academic Publishers, Dordrecht. (1996)
8. Sterling, T.L., Salmon, J., Becker, D.J., Savarese, D.F.: How to Build a Beowulf Cluster: A guide to the Implementation and Application of PC Clusters. The MIT Press, Cambridge London. (1998)
9. Wilkinson, B., Allen, M.: Parallel Programming: Techniques and Applications using Networked Workstations and Parallel Computers. Prentice Hall, New Jersey. (1999)