A Parallel Version for the Propagation Algorithm*

Márcio Bastos Castro Lucas Baldo Luiz Gustavo Fernandes
Mateus Raeder Pedro Velho

Programa de Pós-Graduação em Ciência da Computação, PUCRS
Avenida Ipiranga, 6681 - CEP 90619-900, Porto Alegre, Brazil
{mcastro, lbaldo, gustavo, mraeder, pedro}@inf.pucrs.br

Abstract. This paper presents a parallel version for the Propagation Algorithm which belongs to the region growing family of algorithms. The main goal of our implementation is to decrease de Propagation Algorithm execution time in order to allow its use on image interpolation applications. Our solution is oriented to low cost high performance platforms such as clusters of workstations. Four different input data sets represented by pairs of images were chosen in order to carry out experimental tests. The results obtained show that our parallel version of the Propagation Algorithm presents significant speedups.

1 Introduction

Creating virtual in-between views from two scenes of the same subject taken from different points of view can be a very interesting tool to economize resources in some practical applications [1]. One main example is typically found in teleconferencing with limited network bandwidth. Image-based interpolation is a method to create smooth and realistic virtual views between two original view points. Interpolation applications are usually based on a three-phase algorithm [2]: construction of a dense matching map between the original images, separations of matched areas from unmatched ones and finally the generation of all in-between images. The matching phase is by far the most time consuming one of this procedure. The general technique for matching areas from different images is called region growing. Its basic principle is the use of images characteristics to group neighbor pixels and thus creating regions. In [3], a new region growing algorithm was proposed. It is based on the construction of a quasi-dense matching map between the two original views and it is able to perform more accurate matches. Its originality consists on the adoption of a "best first" strategy to select the next match from a set of seed matches which is updated through the addition of each new found match from the precedent algorithm iteration. This new algorithm was called the Propagation Algorithm, and the improvements on the matching procedure brought together an additional computational cost. This paper proposes a parallel version for the Propagation Algorithm. The target architecture is a cluster of workstations and the implementation was carried out using the standard message passing library MPI [4].

* This work was developed in collaboration with HP Brazil R&D.

The parallelization of the region growing technique has been the subject of several different studies [5]. One of the most spread techniques is based on the "Split and Merge" strategy [6]. On this approach, the merge phase is done through the construction of a non-oriented graph to represent the problem. The graph boundaries are the image regions and the connections between the extremities stand for the neighbors relation of the regions. The first parallel versions of the regions growing algorithm based on the "Split and Merge" approach were implemented over SIMD machines and dynamic structures were used to store image regions information [7, 8]. Another experimental study of the parallel versions of the image segmentation algorithm based on the regions growing technique (also based on the "Split and Merge" approach) was presented by [9]. On this work, the authors propose a new version of the algorithm to determinate the connected components of an image and a new parallel approach is presented for the merge phase.

The paper is organized as follows. In Section 2, the image interpolation application is reviewed, with emphasis to the propagation (region growing) algorithm. After, the proposed parallel approach is described in Section 3. Section 4 presents some experimental results for four different case studies. Finally, some concluding remarks and future directions are given in Section 5.

2 Propagation Algorithm

Before starting the Propagation Algorithm, a preparation phase is necessary to select the seed matches. Points of interest [10] are naturally good seed point candidates because they represent the points of the image that have the highest texture. These points are detected in each separated image. Next, they are matched using the ZNCC (zero-mean normalized cross correlation) measure [3]. At the end of this phase, a set of seed pairs is ready to be used to bootstrap a region growing type algorithm which propagates the matches in the neighborhood of seed points from the most textured pixels to the less textured ones. The Propagation Algorithm itself is based on a classic region growing method for image segmentation [11] which uses pixel homogeneity. However, instead of using pixel homogeneity property, a similar measure based on the matches correlation score is adopted. This propagation strategy could also be justified by the fact that seed pairs are composed by points of interest, which are the local maxima of the texture. Thus, these matches neighbors are also strongly textured what allows good propagation even though they are not local maxima. The neighborhood $N_5(a, A)$ is defined as being all pixels within the 5x5 window centered at these two points (one window per image). For each neighboring pixel in the

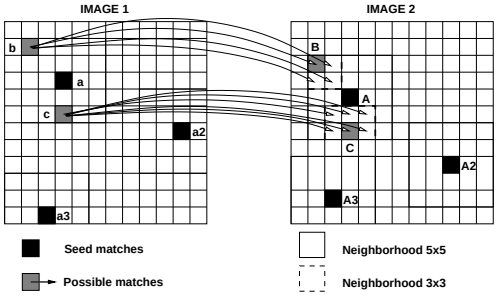


Fig. 1: Neighborhood propagation.

first image, a list of match candidates is constructed. This list consists of all pixels of a 3x3 window in the corresponding neighborhood of the second image (see Fig. 1). The complete definition of the neighborhood $\mathcal{N}(a,\ A)$ is given by:

$$\mathcal{N}(a, A) = \{(b, B), b \in \mathcal{N}_5(a), B \in \mathcal{N}_5(A), (B - A) - (b - a) \in \{-1, 0, 1\}^2\}.$$

The input of the algorithm is a set which contains the current seed pairs. This set is implemented by a heap data structure for a faster selection of the best pair. The output is an injective displacement mapping which contains all the good matches found by the Propagation Algorithm. Briefly, all initial seed pairs are starting points of concurrent propagations. At each step, a match $(a,\ A)$ with the best ZNCC score is removed from the current set of seed pairs. Then, the algorithm looks for new matches in its match neighborhood and, when it finds one, it is added to the current seed pairs set and also to the set of accepted matches which is under construction.

3 Parallel Propagation

The parallel implementation for the Propagation Algorithm discussed on this section was developed in order to allow the use of this new algorithm on realistic situations. Thus, it was necessary to achieve better performances without using parallel programming models oriented to very expensive (but not frequently used) machines. Therefore, the natural choice was a cluster with a message passing programming model.

As seen before, the Propagation Algorithm advances by comparing neighbors pixels through out the source images surface. From some seed pairs, it can form large matching regions on both images surface. In fact, a single seed pair can start a propagation that grows through a large region over the images surface. This freedom of evolution guarantees the algorithm to achieve good results in terms of matched surfaces. Another characteristic is that the algorithm is based on global "best-first" strategy to choose the next seed pair that will start a new propagation, which also has a direct effect on the final match quality. These two characteristics are hard to deal with if one wants to propose a parallel distributed version of the algorithm without loosing quality at the final match. The "best-first" strategy implementation is based on a global knowledge of the seed pairs set, which is not appropriated to a non-shared memory context. In addition, the freedom of

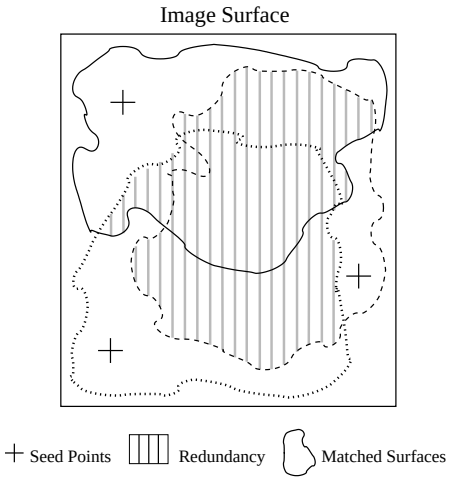


Fig. 2: Redundancy problem.

evolution through out the images surface assumes that the algorithm knows the entire surface of the images, and this can create a situation where several processors are propagating over the same regions at the same time creating a redundancy of computation (Fig. 2). Besides, it is not possible to know in advance how many new matches a seed pair will generate. Thus, from a parallel point of view, the Propagation Algorithm is an irregular and dynamic problem which exhibits unpredictable load fluctuations. Therefore, it requires the use of some load balancing scheme in order to achieve a more efficient parallel solution.

The parallel solution proposed in this paper is based on a master-slave scheme. One processor will be responsible for distributing the work and centralizing the final results. The others processors will be running the Propagation Algorithm, each one using a sub-set of the seed pairs and knowing a pair of corresponding slices over the images surface (coordinates of target slice). The master distributes the seed pairs over the nodes considering their location over the slices. This procedure replaces the global "best-first" strategy by several local "best-first" ones. Each local seed pairs sub-set is still implemented as a heap which is ordered by the pair ZNCC score. This strategy minimizes the problem of loosing quality at the final match.

Once the problem with the global "best-first" strategy is solved, it still remains the problem of the algorithm limitation of evolution over the images surface. As said before, each node can propagate just over the surface of its associated slice in order to avoid computation redundancy. However, forbidding the evolution out of the associated slice generates two kinds of losses. First, some matches are not done because they are just at the border of one slice and one of its points is placed outside it. Second, some regions in one slice may not be reached by any propagation started by a seed pair located inside of its surface, but instead they could be reached by a propagation started at a neighbor slice.

Such a limitation is partially solved by a technique called flexible slices. This technique allows the Propagation Algorithm to expand through the surface of its neighbor slices in a controlled way. As shown on Fig. 3, each processor works over its own associated slice, but it also knows its neighbor slices and it has the permission to propagate over them. But still, it is not interesting to leave the Propagation Algorithm free to compute its neighbors entire surface. This may cause the computation of too many repeated matches. To avoid that, each processor has the permission to compute just over a percentage of its neighbors surface. This percentage is related to the number of slices. A large number of slices implies in thinner slices. In this case, it is acceptable to allow a proces-

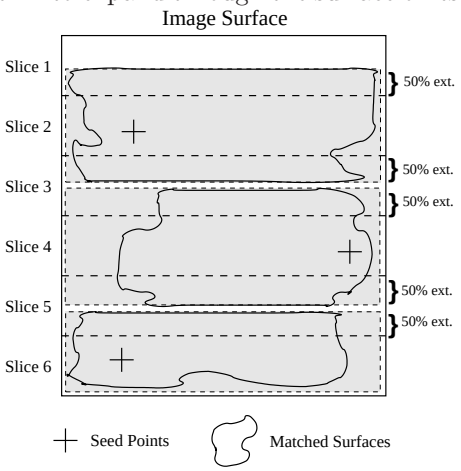


Fig. 3: Flexible slices approach.

sor to advance over a large percentage of its neighbors surfaces. On the other hand, a small number of slices implies in larger slices. Here, the algorithm must not propagate too much over the neighbors surface. Finally, it is important to mention that the master must receive all matches generated by the slaves and it must filter the unavoidable duplicated ones. In order to send these final matches to the master, each slave has a communication buffer which is filled progressively as the Propagation Algorithm advances. When the buffer is full, it is sent to the master. After that, the slave immediately returns to its execution. All slaves do the same procedure, in a way that forces the master to have a receiving queue. This queue is dimensioned to avoid buffer losses by the master. When a slave reaches the end of its seed pairs sub-set, it sends an incomplete buffer to the master. When the master receives an incomplete buffer, it knows that the sender has finished its work and sends a new slice (seed pairs sub-set) back to it (if there is still sub-sets available). Figure 4 shows the complete flow-chart for the parallel Propagation Algorithm.

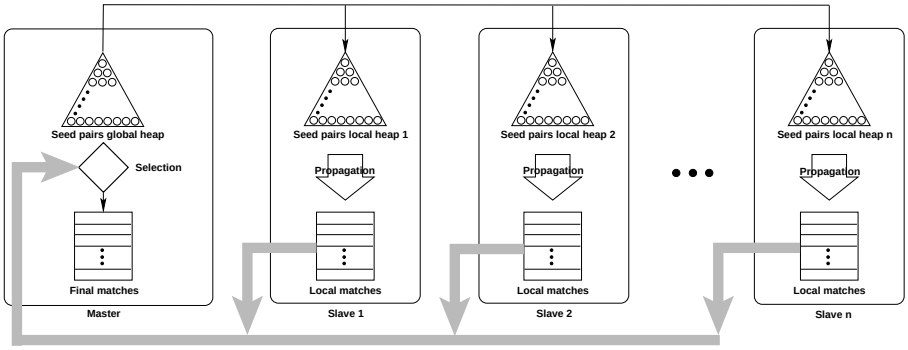


Fig. 4. Flow-chart of the parallel Propagation Algorithm.

The last problem to deal with in the parallelization of the Propagation Algorithm is the workload distribution. If the source images are divided into more slices than the number of nodes available, the following strategy is adopted:

1. the master divides the set of seed pairs into sub-sets based on their location over the slices;
2. each slave receives one slice with its associated sub-set;
3. each slave computes its own sub-set of seed pairs;
4. when there is no more seed pairs to compute, the slave sends a signal to the master;
5. if there is some available slices remaining, the master choose a new one and send it to the available slave.

In fact, the master has a queue of slices, organized by their position over the images surface. In order to choose which slice will be sent to an available

slave, the master just gets the first slice of this queue. This procedure is sufficient to avoid the workload unbalance problem originated by the different amount of seed matches each slice has.

4 Experimental Results

In order to perform the experimental tests of the parallel implementation of the Propagation Algorithm, four case studies were selected. Table 1 presents the size of the images that compose those case studies with their respective sequential execution times obtained using a Pentium III 1 Ghz with 256 MB RAM.

Table 1. Execution times for the sequential Propagation Algorithm.

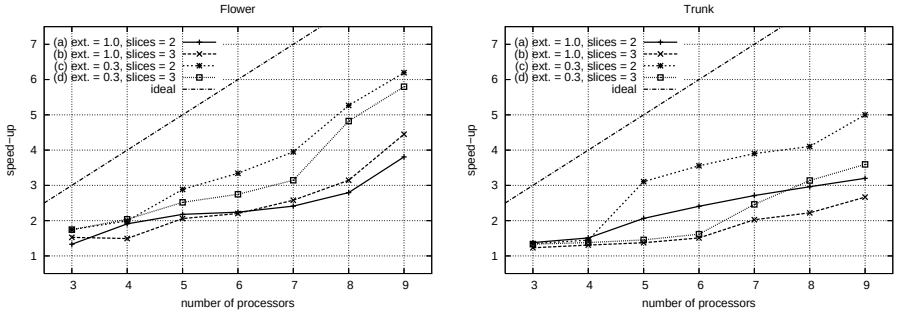
Image	Flower	House	Rock	Trunk
Size (pxs)	368x384	768x512	512x768	360x240
Propagation time (s)	6.32	15.24	14.32	3.20

Each pair of images shows specific characteristics. The Flower pair is the only one based on non-realistic images. Both, the House and the Rock pairs have the same size, but the House pair has more textured regions and presents occluded elements. Finally, the Trunk pair is the only one based on a gray scale of colors and it has the smallest number of textured regions. This set of input images is clearly not exhaustive, but the pairs of images were carefully chosen to make it possible to verify the the parallel Propagation Algorithm behavior on different situations.

For all input images, experimental tests were carried out varying on the number of processors[1](N), number of slices per slave (fine grain and coarse grain) and the redundancy extension allowed over the slices. The number of slices per slaves is obtained by $2 \times N$ (coarse grain) and by $3 \times N$ (fine grain). Moreover, the slices redundancy extension used was 30% and 100% of the slices height. Figure 5 shows the speedup, execution time (T) and efficiency (E) of the parallel Propagation Algorithm for each case study. The experimental tests showed that, for all input images pairs, our parallel implementation achieved an execution time reduction about 81% ($\simeq$ 79.26% for the Rock, $\simeq$ 80.01% for the Trunk, $\simeq$ 81.49% for the House and $\simeq$ 83.86 for the Flower) using 9 processors. On the other hand, all executions carried out with more than 9 processors presented a significant lost of performance.

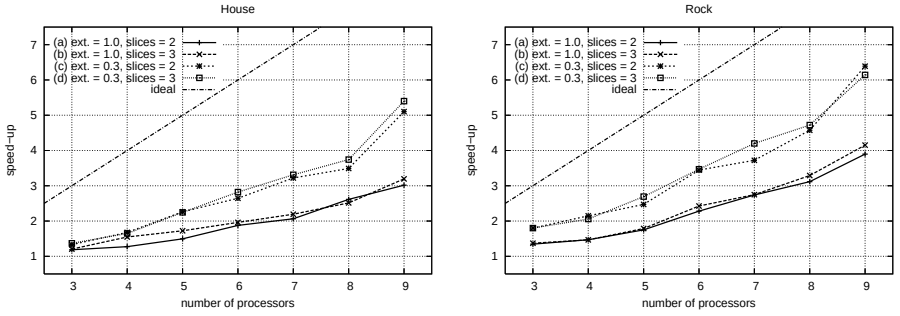
The analysis of the curves on the graphs of Fig. 5, one can clearly identify that the 30% of redundancy extension always results in a better efficiency. This result was expected, since with a lower redundancy allowed there are less pairs to match. We could then expect even better results with less than 30% extension, however this is not possible due to the lost of matches at boundaries of each slice what compromises the final match quality.

[1] The target architecture was a cluster with 8 nodes Pentium III 1 Ghz dual and 256 MB RAM connected by a 100 Mb Fast-Ethernet network.



		number of processors						
		3	4	5	6	7	8	9
(a)	T	4.75	3.31	2.90	2.82	2.62	2.26	1.66
	E	44.35	47.73	43.59	37.35	34.46	34.96	42.30
(b)	T	4.14	4.23	3.07	2.87	2.45	2.01	1.42
	E	50.89	37.35	41.17	36.70	36.85	39.30	49.45
(c)	T	3.62	3.17	2.19	1.89	1.60	1.20	1.02
	E	58.20	49.84	57.72	55.73	56.43	65.83	68.85
(d)	T	3.62	3.10	2.51	2.30	2.01	1.31	1.09
	E	58.20	50.97	50.36	45.80	44.92	60.31	64.42

		number of processors						
		3	4	5	6	7	8	9
(a)	T	2.32	2.12	1.55	1.33	1.18	1.08	1.00
	E	45.98	37.74	41.29	40.10	38.74	37.04	35.56
(b)	T	2.60	2.45	2.33	2.12	1.58	1.44	1.20
	E	41.03	32.65	27.47	25.16	28.93	27.78	29.63
(c)	T	2.37	2.21	1.03	0.90	0.82	0.78	0.64
	E	45.01	36.20	62.14	59.26	55.75	51.28	55.56
(d)	T	2.40	2.32	2.20	1.98	1.30	1.02	0.89
	E	44.44	34.48	29.09	26.94	35.16	39.22	39.95

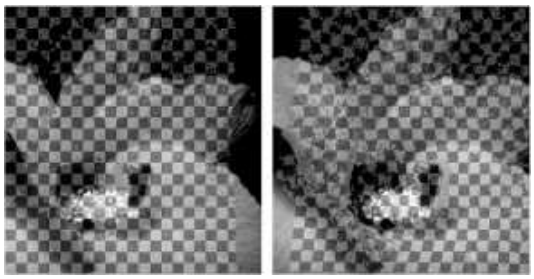


		number of processors						
		3	4	5	6	7	8	9
(a)	T	12.84	11.96	10.20	8.12	7.38	5.83	5.05
	E	39.56	31.86	29.88	31.28	29.50	32.68	33.53
(b)	T	12.66	9.85	8.84	7.77	6.95	6.06	4.77
	E	40.13	38.68	34.48	32.69	31.33	31.44	35.50
(c)	T	11.44	9.08	6.74	5.76	4.74	4.37	2.99
	E	44.41	41.96	45.22	44.10	45.93	43.59	56.63
(d)	T	11.13	9.25	6.79	5.40	4.61	4.08	2.82
	E	45.64	41.19	44.89	47.04	47.23	46.69	60.05

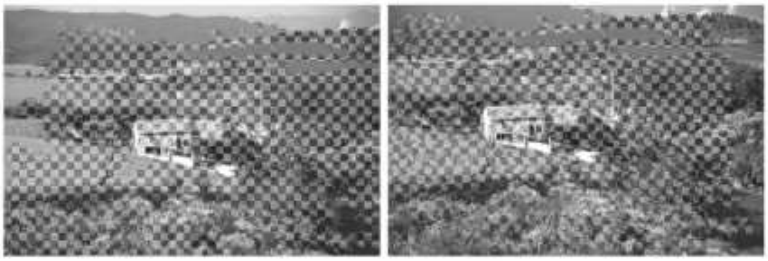
		number of processors						
		3	4	5	6	7	8	9
(a)	T	14.10	12.91	10.85	8.32	6.93	6.09	4.87
	E	33.85	27.73	26.40	28.69	29.52	29.39	32.67
(b)	T	13.84	12.95	10.62	7.82	6.91	5.77	4.57
	E	34.49	27.64	26.97	30.52	29.61	31.02	34.82
(c)	T	10.54	8.84	7.68	5.51	5.10	4.15	2.97
	E	45.29	40.50	37.29	43.32	40.11	43.13	53.57
(d)	T	10.55	9.23	7.05	5.47	4.52	4.02	3.09
	E	45.24	38.79	40.62	43.63	45.26	44.53	51.49

Fig. 5. Results: speedup, execution time (T, in seconds) and efficiency (E, in %).

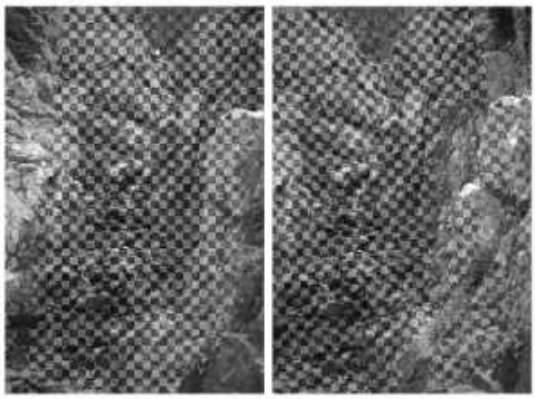
Examples of the parallel Propagation Algorithm output for each case study ((a) Flower, (b) House, (c) Rock and (d) Trunk) can be visualized at Fig.6. The squared regions in both images of each pair show the extension of the matched regions obtained from the seed matches. Readers can notice that the Propagation Algorithm advances better over the textured surfaces. Regions like the sky in the Trunk pair or the grass in the House pair were not matched due to absence of texture. Furthermore, some regions on the images boundaries cannot be matched because they do not appear in both views.



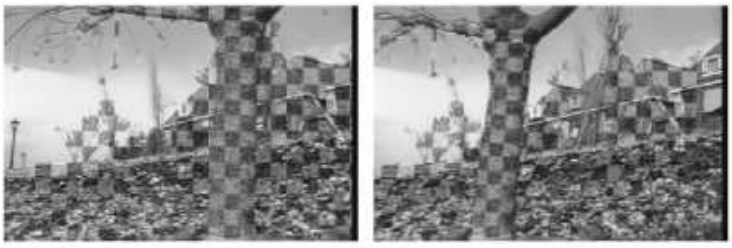
(a) Flower



(b) House



(c) Rock



(d) Trunk

Fig. 6. Output of the parallel Propagation Algorithm for each case study.

5 Conclusions

The implementation of a parallel version for the Propagation Algorithm was presented in this paper. The particularity of this algorithm consists on the adoption of a "best first" strategy to select the next match from a set of seed matches firing several propagations that can advance over the same images zones generating a large redundancy in the computation of the seed matches. Our parallel version is based on a master/slave scheme and we proposed a new technique called flexible slices to solve the redundancy problem. Several experiments were carried out in order to verify the usability of our approach and the results present a significant gain of performance. Finally, it is the authors opinion that the work developed so far was worthwhile. The results obtained are interesting and the implementation allowed a quite good understanding of the problem, leading to promising directions for further investigations.

References

1. Seitz, S., Dyer, C.: Physically-Valid View Synthesis by Image Interpolation. In: Proceedings of the International Workshop on Representations of Visual Scenes, Cambridge, Massachussets, USA (1995) 26–33
2. Lhuillier, M., Quan, L.: Image Interpolation by Joint View Triangulation. In: Proceedings of the International Conference on Computer Vision and Pattern Recognition, Fort Collins, Colorado, USA (1999) 139–145
3. Lhuillier, M., Quan, L.: Robust Dense Matching using Local and Global Geometric Constraints. In: Proceedings of the 15th International Conference on Pattern Recognition. (2000) 968–972
4. Snir, M., Otto, S., Huss-Lederman, S., Walker, D., Dongarra, J.: MPI: the complete reference. MIT Press (1996)
5. Alnuweiri, H., Prasanna, V.: Parallel architectures and algorithms for image component labeling. IEEE Trans. Patt. Anal. Machine Intell. **14** (1992) pp. 1014–1034
6. Horowitz, S., Pavlidis, T.: Picture segmentation by a directed split-and-merge procedure. In: Proceedings of the 2nd International Joint Conference on Pattern Recognition. (1974) pp. 424–433
7. Tilton, J.: Image segmentation by iterative parallel region growing with applications to data compression and image analysis. In: Proceedings of the 2nd Symposium on the Frontiers of Massively Parallel Computation. (1988) pp. 357–360
8. Willebeck-LeMair, M., Reeves, A.: Solving non-uniform problems on simd computers: case study on region growing. Journal of Paralle and Distributed Computing **8** (1990) pp. 135–149
9. Jájá, J., Bader, D., Harwood, D., Davis, L.: Parallel algorithms for image enhancement and segmentation by region growing with an experimental study. Technical report, Institute for Advanced Computer Studies, University of Maryland (1995)
10. Schimid, C., Mohr, R., Bauckhage, C.: Comparing and Evaluating Interest Points. In: Proceedings of the 6th International Conference on Computer Vision, Bombay, India (1998) 230–235
11. Monga, O.: An Optimal Region Growing Algorithm for Image Segmentation. International Journal of Pattern Recognition and Artificial Intelligence **1** (1987) 351–375