A Parallel Solution in Texture Analysis Employing a Massively Parallel Processor

Andreas I. Svolos, Charalambos Konstantopoulos, and Christos Kaklamanis

Computer Technology Institute and Computer Engineering & Informatics Dept.,
Univ. of Patras, GR 265 00 Patras, Greece,
svolos@cti.gr

Abstract. Texture is a fundamental feature for image analysis, classification, and segmentation. Therefore, the reduction of the time needed for its description in a real application environment is an important objective. In this paper, a texture description algorithm running over a hypercube massively parallel processor, is presented and evaluated through its application in real texture analysis. It is also shown that its hardware requirements can be tolerated by modern VLSI technology.

Key words: texture analysis, co-occurrence matrix, hypercube, massively parallel processor

1 Introduction

Texture is an essential feature that can be employed in the analysis of images in several ways, e.g. in the classification of medical images into normal and abnormal tissue, in the segmentation of scenes into distinct objects and regions, and in the estimation of the three-dimensional orientation of a surface. Two major texture analysis methods exist: statistical and syntactic or structural. Statistical methods employ scalar measurements (features) computed from the image data that characterize the analyzed texture. One of the most significant statistical texture analysis methods is the Spatial Gray Level Dependence Method (SGLDM). SGLDM is based on the assumption that texture information is contained in the overall spatial relationship that the gray levels have to one another. Actually, this method characterizes the texture in an image region by means of features derived from the spatial distribution of pairs of gray levels (second-order distribution) having certain inter-pixel distances (separations) and orientations [1].

Many comparison studies have shown SGLDM to be one of the most significant texture analysis methods [2]. The importance of this method has been shown through its many applications, e.g. in medical image processing [3]. However, the co-occurrence matrix [1], which is used for storing the textural information extracted from the analyzed image, is inefficient in terms of the time needed for its computation. This disadvantage limits its applicability in real-time applications and prevents the extraction of all the texture information that can be captured

B. Monien and R. Feldmann (Eds.): Euro-Par 2002, LNCS 2400, pp. 431–435.

by the SGLDM. The parallel computation of the co-occurrence matrix is a potential solution to the computational time inefficiency of this data structure. The first attempt to parallelize this computation was made in [4]. However, to the best of our knowledge, the only previous research effort of parallelization using a massively parallel processor was made in [5]. The reason is that until recently, full parallelization was possible only on very expensive machines. The cost of the parallel computation was prohibitive in most practical cases. For this reason, even in [5], there was a compromise between hardware cost and computational speed. The parallel co-occurrence matrix computation ran over a Batcher network topology. However, this parallel scheme had two significant drawbacks. First, the Batcher network requires a very large number of processing elements and interconnection links limiting its usefulness to the analysis of very small image regions. Second, the parallel algorithm proposed in [5] assumes an off-line pre-computation of the pairs of pixels that satisfy a given displacement vector in each analyzed region. This pre-computation has to be performed by another machine, since the Batcher network does not have this capability.

The rapid evolution of CMOS VLSI technology allows a large number of processing elements to be put on a single chip surface [6], dramatically reducing the hardware cost of the parallel implementation. Employing a regular parallel architecture also helps towards achieving a larger scale of integration. In this paper, a parallel algorithm for the computation of the co-occurrence matrix running on a hypercube massively parallel processor, is presented and evaluated through its application to the analysis of real textures.

2 The Parallel Algorithm for the Co-occurrence Matrix Computation

The processing elements of the massively parallel processor employed in this paper are interconnected via a hypercube network. The hypercube is a general-purpose network proven to be efficient in a large number of applications, especially in image processing (2D-FFT, Binary Morphology) [7]. It has the ability to efficiently compute the gray level pairs in an analyzed image region for any displacement vector. Moreover, its large regularity makes feasible the VLSI implementation of this parallel architecture.

In this paper, a modified odd-even-merge sort algorithm is employed for the parallel computation of the co-occurrence matrix. In the proposed algorithm, each element is associated with a counter and a mark bit. The counter gives the number of times an element has been compared with an equal element up to the current point of execution. The mark bit shows whether this element is active, i.e. it participates in the parallel computation (bit = 0), or is inactive (bit = 1). Each time two equal elements are compared, the associated counter of one of these two elements increases by the number stored in the counter of the other element. Also, the mark bit of the other element becomes 1, that is, the element becomes inactive. Inactive elements are considered to be larger than the largest element in the list. In the case that the compared elements are not equal,

```
for i := 1 to m do
    for j := 1 to i − 1 do                    /* transposition sub-steps */
        parbegin
            P^1 = P_m P_{m−1} ... P_{i+1} 1 P_{i−1} ... P_{j+1} 0 P_{j−1} ... P_1;
            P^2 = P_m P_{m−1} ... P_{i+1} 1 P_{i−1} ... P_{j+1} 1 P_{j−1} ... P_1;
            P^1 ↔ P^2;
        parend
    od
    for j := i to 1 do                        /* comparison sub-steps */
        parbegin
            P^1 = P_m P_{m−1} ... P_{j+1} 0 P_{j−1} ... P_1;
            P^2 = P_m P_{m−1} ... P_{j+1} 1 P_{j−1} ... P_1;
            P^2 → P^1;                        /* the content of element P^2 is transferred to element P^1 */
            if P^1.M == 0 and P^2.M == 0 and P^1.(A_1, B_1) == P^2.(A_2, B_2) then
                P^1.C := P^1.C + P^2.C;
                P^2.M := 1;
                P^1 → P^2;                    /* the updated content of P^2 is sent back to P^2 */
            else if P^1.M == 1 or P^1.(A_1, B_1) > P^2.(A_2, B_2) then
                P^1 → P^2;                    /* P^2 gets the content of P^1 */
                P^1 := P^2;                   /* P^1 gets the content sent from P^2 */
            else
                nop;
            endif
        parend
    od
od
```

Fig. 1. The pseudocode of the proposed parallel algorithm for the co-occurrence matrix computation

the classical odd-even-merge sort algorithm is applied. At the end, the modified algorithm gives for each active element its times of repetition in the initial list. If each element in the list is a pair of gray levels in the analyzed region that satisfies a given displacement vector, it is straightforward to see that the above algorithm eventually computes the corresponding co-occurrence matrix.

The pseudocode of the algorithm is shown in Fig. 1. In this figure, the language construct **parbegin. . .parend** encloses the instructions, which are executed by all processing elements, concurrently. The " = " operator declares equivalence of notations. Actually, the right operand is the binary representation of processing element P in the hypercube. The " ↔ " operator performs a transposition of the contents of its operands (processing elements) through the hypercube network. The "→" operator transfers data from its left operand to its right operand over the hypercube network. Finally, $P.(A, B)$ is the pair of gray levels stored in processing element P, $P.C$ is the counter associated with gray level pair (A, B) and $P.M$ is the corresponding mark bit.

3 Results and Discussion

In order to show the time performance of the proposed parallel algorithm in a practical case, a large number of samples from natural textures were analyzed employing the SGLDM (fur, water, weave, asphalt, and grass) [8]. The co-occurrence matrices were computed using the proposed parallel algorithm running on the hypercube, the algorithm running on the Batcher network and

the fastest serial algorithm. Each image had a dynamic range of 8 bits (256 gray levels). From each image, data sets of 64 non-overlapping sub-images of size 64×64, 256 non-overlapping sub-images of size 32×32, and 1024 non-overlapping sub-images of size 16×16 were extracted. 8 displacement vectors were employed in the texture analysis of all five categories of samples, namely (1,0), (0,1), (1,1), (1,-1), (2,0), (0,2), (2,2), and (2,-2). In this experiment, both parallel architectures (hypercube and Batcher network) were assumed to be consisted of all processing elements required to fully take advantage of the parallelism inherent in the co-occurrence matrix computation for a specific image size. The compared architectures were simulated on the Parallaxis simulator [9]. The total computational time from the analysis of all images in each of the 15 data sets was estimated. Then, an averaging of the computational time over all data sets corresponding to the same image size was performed. The estimated average times were employed in the computation of the speedups. Fig. 2 a) shows the speedup of the hypercube over the serial processor whereas Fig. 2 b) shows the speedup of the hypercube over the Batcher network.

The hypercube attains a greater speedup in all compared cases (see Fig. 2). From Fig. 2 a), it is clear that the speedup increases as the size of the analyzed images increases. It becomes about 2183 for the analyzed sets of the 64×64 images. The reason for this increase is that the proposed algorithm running on the hypercube can fully utilize the inherent parallelism in co-occurrence matrix computation. As we increased the number of processing elements in the performed experiment to handle the larger image size the proposed parallel algorithm became much faster than the serial one. This phenomenon also appears in Fig. 2 b), where the speedup rises from about 6, in the case of the 16×16 images, to about 30, in the case of the 64×64 images. From this figure, it is obvious that in all analyzed cases the hypercube network was superior to the Batcher network. However, in this performance comparison the achieved speedup was mainly due to the efficient way of deriving the gray level pairs for a given displacement vector employing the proposed architecture.

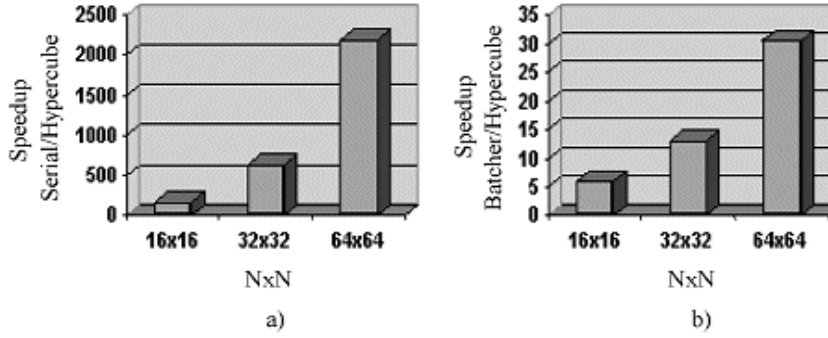

Fig. 2. a) The speedup of the hypercube over the serial processor for various image sizes. b) The speedup of the hypercube over the Batcher network for various image sizes

Even though the degree of the hypercube increases logarithmically with the number of nodes, which is actually its biggest disadvantage, the rapid evolution of the VLSI technology and the large regularity of this type of architecture made possible the manufacturing of large hypercubes. With the current submicron CMOS technology [6], hundreds of simple processing elements can be put on a single chip allowing the implementation of a massively parallel system on a single printed circuit board for the simultaneous processing of the pixels of a 64×64 gray level image with a dynamic range of 8 bits (256 gray levels). Moreover, from the pseudocode in Fig. 1, it is clear that the structure of each processing element in the proposed parallel architecture can be very simple.

4 Conclusions

The parallel algorithm for the SGLDM proposed in this paper was shown to be superior in all compared cases, in terms of computational time. The analysis of real textures showed that the algorithm has the ability to fully exploit the parallelism inherent in this computation. Furthermore, the employed parallel architecture needs much less hardware than the previously proposed massively parallel processors, which can be tolerated by modern VLSI technology.

Acknowledgements

This work was supported in part by the European Union under IST FET Project ALCOM-FT and Improving RTN Project ARACNE.

References

1. Haralick, R., Shanmugam, K., Dinstein, I.: Textural features for image classification. IEEE Trans. Syst. Man. Cybern. **SMC-3** (1973) 610–621
2. Ohanian, P., Dubes, R.: Performance evaluation for four classes of textural features. Patt. Rec. **25** (1992) 819–833
3. Kovalev, V., Kruggel, F., et al.: Three-dimensional texture analysis of MRI brain datasets. IEEE Trans. Med. Imag. **MI-20** (2001) 424–433
4. Kushner, T., Wu, A., Rosenfeld, A.: Image processing on ZMOB. IEEE Trans. on Computers **C-31** (1982) 943–951
5. Khalaf, S., El-Gabali, M., Abdelguerfi, M.: A parallel architecture for co-occurrence matrix computation. In Proc. 36th Midwest Symposium on Circuits and Systems (1993) 945–948
6. Ikenaga, T., Ogura, T.: CAM^2: A highly-parallel two-dimensional cellular automaton architecture. IEEE Trans. on Computers **C-47** (1998) 788–801
7. Svolos, A., Konstantopoulos, C., Kaklamanis, C.: Efficient binary morphological algorithms on a massively parallel processor. In IEEE Proc. 14th Int. PDPS. Cancun, Mexico (2000) 281–286
8. Brodatz, P.: Textures: a Photographic Album for Artists and Designers. Dover Publ. (1966)
9. http://www.informatik.uni-stuttgart.de/ipvr/bv/p3