Parallelization of symmetry detection algorithms on a network of workstations

R. Parthiban[a,*], C.P. Ravikumar[b], R. Kakarala[c], J. Sivaswamy[c]

[a]*Pertech Computer Limited, Bangalore, India*
[b]*Department of Electrical Engineering, Indian Institute of Technology, New Delhi 110016, India*
[c]*Department of Electrical and Electronic Engineering, University of Auckland, Private Bag, Auckland, New Zealand*

Received 3 December 1995; accepted 16 September 1996

Abstract

Detection of spatial symmetry is useful in several computer vision applications. Due to the real-time nature of the applications, it is important that symmetry detection algorithms be computationally efficient. Sequential algorithms for finding various types of planar symmetries in images are CPU-intensive, prompting us to look for fast parallel implementations. In this paper, we propose parallel algorithms for symmetry detection, and report an implementation on a distributed computing environment consisting of a network of Sun workstations. Experiments revealed close-to-linear speedup on a number of test images which we considered.

Keywords: Image symmetry detection; Parallel virtual machine; Parallel image processing

1. Introduction

A principal goal of computer vision research is to develop techniques that can identify from images various properties of objects, such as convexity, symmetry, and separability into parts. Symmetry identification, which is the focus of this paper, is particularly important in several computer vision applications, e.g. following cars at a safe distance [1], determining where human faces occur in images [2], and segmenting images into regions of different texture [3]. We consider the first application [1] in more detail to demonstrate the utility of symmetry information. Most modern cars, when seen from directly behind, are approximately symmetric around a vertical axis passing midway between the ends of the rear bumper. In order to provide a "driver's aid" that automatically monitors the movements of the car directly in front of the driver, a computer vision system is employed to detect if there exists a vertical axis of symmetry in video images of the road ahead. If such an axis exists, the vision system tracks the changes in the axis resulting from movements of the car directly in front. This information is used by automatic road-following and accident-avoidance equipment [4].

Two-dimensional spatial symmetry can be rotational, reflectional, or translational. In the first two computer vision applications mentioned above, it is crucial to accurately locate the axis of reflectional symmetry, i.e., the line about which one side is the mirror image of the other. In the third application, it is also important to determine whether translational or rotational symmetries exist in addition to reflectional symmetries.

This paper is concerned with the design and implementation of efficient algorithms for detecting rotational and reflectional symmetries in images. Serial algorithms for detecting these symmetries have been proposed in the literature [1,3,5]. Unfortunately, these algorithms are computationally demanding (typically requiring over 30 minutes to process a single image) and do not lend themselves to applications such as [4], where processing at video-rate (30 images/second) is required. The aim of this paper is to investigate parallel formulations of symmetry detection algorithms, a topic that to our knowledge has not been studied previously. In particular, we focus on some algorithms that use simple linear operators which have been proposed recently [5]. We believe that these algorithms have particularly good

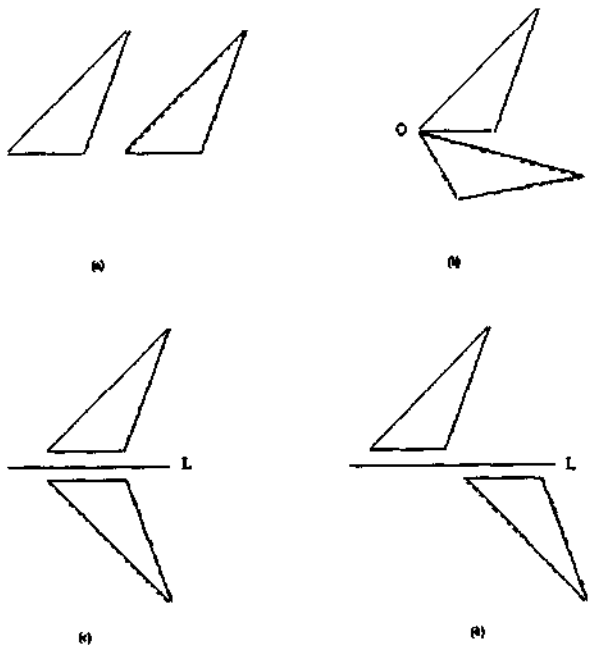

Fig. 1. Figure indicating the result of various transforms: (a) translation; (b) rotation (c) reflection, and (d) glide reflection.

prospects for parallelization, although they are not necessarily unique in that respect. Our main result is a method for parallelization for the symmetry detection algorithms proposed in [5] which enables close-to-linear speedup when multiple processors are used. The implementation of this method on a network of workstations is described and test results are provided on several images.

The paper is organized as follows. Basic concepts and principles of planar symmetries are described in Section 2. In Section 3, we review the linear filter-based algorithms for symmetry detection. In Section 4, the parallel versions of these algorithms are discussed and experimental results are reported.

2. Theory of symmetry detection

As shown in Fig. 1, exactly four types of geometric transformation preserve the lengths of objects in the plane, translations, rotations, reflections, and glide reflections[1] [6]. An image is said to be symmetric if one of the four transformations can be applied to it without changing the appearance of the image. In this section, we develop the theoretical basis for detecting translational, rotational, and reflectional symmetries.

Formally, an image f is symmetric with respect to a translation vector T if $f(x) = f(x+T)$. Such an image must also be periodic with period T. Periodicity may be detected from the autocorrelation of the image, and the value of T estimated by measuring the displacements between autocorrelation peaks. Since this technique is well-known [7], we proceed to examine methods to detect other symmetries.

Let x be any point in the domain of the image. The notation $f_x(r,\theta)$ represents the image expressed in polar coordinates in a neighborhood about x. For $k > 0$, an image is said to possess rotational symmetry of order k about x if

$$f_x\left(r, \theta + \frac{2\pi}{k}\right) = f_x(r,\theta). \tag{1}$$

If in addition an image is also periodic, i.e., it has translational symmetry, then it is a remarkable fact, known as the crystallographic restriction, that $k \leq 2$ if the pattern is periodic in one direction, and that $k = 1, 2, 3, 4$ or 6 (but not 5) if the pattern is periodic in two directions [8].

Let θ_0 be the inclination, measured counterclockwise from the horizontal, of a line L passing through x. An image is said to be reflectionally symmetric across L if

$$f_x(r, \theta_0 + \theta) = f_x(r, \theta_0 - \theta). \tag{2}$$

To detect either rotational or reflectional symmetry, we use the method of angular Fourier series. In this method, the circular neighborhood of every pixel x is expanded in a Fourier series in the angular coordinate. Specifically, we write

$$f_x(r,\theta) = \sum_{n=-\infty}^{\infty} F_x(r,n) e^{jn\theta}. \tag{3}$$

Here, $F_x(r,n)$ is the Fourier series coefficient computed as follows:

$$F_x(r,n) = \frac{1}{2\pi}\int_0^{2\pi} f_x(r,\theta) e^{-jn\theta} d\theta. \tag{4}$$

Coefficients that are measured in this way vary with radius r. In order to establish whether some symmetry exists in some suitably large neighborhood of x, it is useful to average the coefficients over all radii up to some chosen maximum value R. This average is calculated as follows:

$$F_x(n) = \frac{1}{R}\int_0^R F_x(r,n) r dr. \tag{5}$$

Putting Eqs. (4) and (5) together, we have that

$$F_x(n) = \frac{1}{2\pi R}\int_o^{2\pi}\int_0^R f_x(r,\theta) e^{-jn\theta} r d\theta dr. \tag{6}$$

The average angular Fourier series coefficients have the following properties that are of importance for the proposed symmetry tests.

1. Rotation of f is equivalent to a phase shift of F_x. If $f_x(r,\theta) \rightarrow f_x(r,\theta+\theta_0)$ (representing a counterclockwise rotation by θ_0), then the corresponding

[1] A glide reflection is a translation simultaneous with reflection along the same axis.

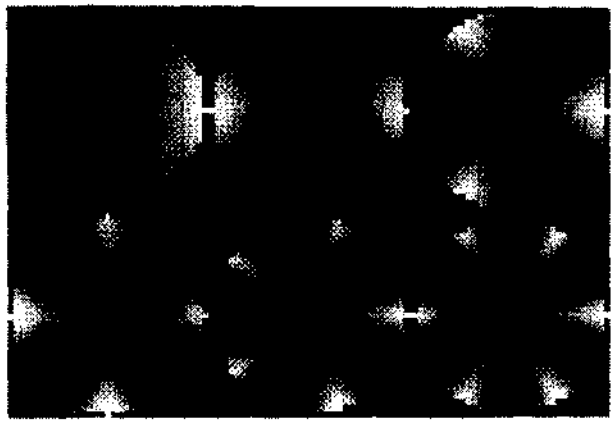

Fig. 2. The real parts of the point-spread functions for the six filters $h_1, \ldots, h_6$ are shown.

average angular Fourier coefficients undergo the transformation

$$F_x(n) \rightarrow F_x(n)e^{jn\theta_0}. \tag{7}$$

2. Reflection of f_x about any axis passing through x and making an angle θ_0 with the horizontal is equivalent to conjugation followed by a phase shift. To establish this, let f'_x denote the result of applying such a reflection to f_x. Then we have $f'_x(r,\theta) = f_x(r, 2\theta_0 - \theta)$ for all angles θ, as is easily verified by checking that $f'_x(r, 2\theta_0) = f_x(r,\theta)$ and that $f'_x(r,\theta_0) = f_x(r,\theta_0)$. We obtain from this that

$$F'_x(n) = F^*_x(n)e^{jn2\theta_0}, \tag{8}$$

where * denotes complex-conjugate.

Using the average angular Fourier series coefficients, specific tests may be formulated for rotational and reflectional symmetry. In particular, an image contains rotational symmetry of order k about x if and only if $f_x(r,\theta) = f_x(r, \theta + \frac{2\pi}{k})$ for all θ, or equivalently by (7) we have,

$$F_x(n) = F_x(n)e^{jn\frac{2\pi}{k}}, \quad \forall n. \tag{9}$$

For this to happen, it must be true that $F_x(n) = 0$ for all n not divisible by k. Furthermore, an image is reflectionally symmetric across a line L passing through x if and only if we have

$$F_x(n) = F^*_x(n)e^{j2n\theta_0}, \quad \forall n. \tag{10}$$

This equation is satisfied if and only if the coefficients F_x have linear phase, which means that for appropriate real numbers $R_x(n)$ we must have:

$$F_x(n) = R_x(n)e^{jn\theta_0}, \quad \forall n. \tag{11}$$

3. Algorithms for symmetry detection

The local Fourier series coefficients in Eq. (6) may be computed at every location in a digital image by formulating the computation as a convolution. Specifically, for any integer $n > 0$ we define the impulse response of a 2-D filter h_n as follows:

$$h_n(x) = \begin{cases} \frac{1}{R}e^{-jn\arg x} & \text{if } \|x\| \le R, \\ 0 & \text{otherwise.} \end{cases} \tag{12}$$

Here, "arg x" stands for the angle made by the vector x with respect to the horizontal, and $\|x\|$ is the length of x. The resulting response lies within a rectangle of dimensions $(2R+1) \times (2R+1)$ in the plane. The algorithms that we describe below use only the first six filters $h_1, h_2, \cdots, h_6$. More filters can of course be used, but the additional computation did not make any significant improvements in our experiments as described in Section 4. Interestingly, six is maximum number of filters necessary for the special case of periodic images, since the crystallographic restriction limits the order of rotational symmetry to a maximum of six in this case. Fig. 2 shows the point-spread functions of the six filters.

From Eq. (6), it may be seen that the response of a filter h_n to the image at any location x gives a good approximation to the n-th average Fourier series coefficient:

$$F_x(n) \approx \sum_{\|y\| \le R} f_x(y)h_n(y). \tag{13}$$

(The summation is evaluated in a $(2R+1) \times (2R+1)$ neighborhood of pixel locations y which is centered at x). When the input image is of dimension $N \times M$, the computation is performed without using the boundaries to produce an output image of size $(N-2R) \times (M-2R)$.

The rotational symmetry test is implemented as follows. To detect k-th order symmetry for $k \le 6$, we compute the Fourier coefficients $F_x(1)$, $F_x(2)$, $\cdots$, $F_x(6)$, and at every output pixel form the ratio

$$\frac{\sum_{n:k|n} |F_x(n)|}{\sum_{n=1}^{6} |F_x(n)|}. \tag{14}$$

(Here, the notation $n : k|n$ in the numerator means that the sum is taken over all n divisible by k.) This ratio always lies between 0 and 1, with 0 signifying lack of symmetry and 1 signifying the presence of symmetry. The pixel locations of high rotational symmetry are then identified by comparing the output pixel values to a fixed threshold T; the value of $T = 0.95$ was found to work well in the experiments described below.

The reflectional symmetry test is implemented by using a similar ratio. It may be seen from Eq. (11) that if the coefficients F_x have the linear phase property (11), then the following quantity, called the bispectrum, is

always real-valued for all n, m:

$$F_x(n)F_x(m)F_x^*(n+m). \tag{15}$$

To see this, note that the sum of phases of $F_x(n)$ and $F_x(m)$ respectively is cancelled by the conjugate term $F_x^*(n+m)$. Among the attractive properties of the bispectrum is that it is unbiased (that is, on average, the bispectrum does not change its value) in a Gaussian noise background, which is important for detecting reflection symmetries in noisy images [9]. The unbiasedness is essentially due to the product of three terms being used, which acts like a cubic function $Y = X^3$, allowing the sign of the input X to remain in the output Y.

The bispectrum may be used to detect reflectional symmetry by forming at every pixel x the ratio

$$\frac{\sum_{n,m=1}^{K} |\Re\{F_x(n)F_x(m)F_x^*(n+m)\}|}{\sum_{n,m=1}^{K} |F_x(n)F_x(m)F_x^*(n+m)|}. \tag{16}$$

Here $\Re\{\cdot\}$ denotes the real part of the complex number, and K is the maximum frequency computed (in our experiments, $K = 6$). As in the case of rotational test, this ratio lies between 0 and 1, with 0 signifying lack of symmetry and 1, perfect symmetry. A thresholding scheme similar to that previously described is used to identify pixel locations of high reflection symmetry.

4. Parallel algorithm

The methods described in the previous section work by computing the local Fourier coefficients $F_x(1), \ldots, F_x(6)$ at each location x, and subsequently computing the relevant statistic, either (14) or (16). If only a single processor is available in a computer system implementing these tests, there is no alternative but to calculate one coefficient after another at each x. This sequential approach is slow, making a real-time implementation all but impossible. However, the algorithm itself appears to be well-suited to a parallel implementation, a possibility that we investigate in this section.

Parallel architectures and algorithms for image processing and image understanding tasks have been studied in the literature [10]. A number of special-purpose architectures and algorithms based on SIMD and systolic array style of computing have been proposed for low-level computer vision tasks. For image understanding problems such as object recognition, MIMD style computing appears to be more suitable [11]. In fact, experts in the area of computer vision now concur on the notion that heterogeneous computing is best suited for applications in machine vision, since different classes of vision algorithms call for different styles of computing [12]. In the recent past, networks of workstations have become attractive as vehicles for inexpensive parallel computing. With greater availability of workstations and high-speed inter-connections, it has become possible to efficiently utilize the idle cycles of individual machines in a network to carry out useful computation. For example, Lee and Hamdi describe parallel image processing applications on a network of workstations under the EXPRESS programming environment [13]. In our work, a distributed computing environment called PVM (Parallel Virtual Machine) was used to distribute computations across the workstation [15,14].

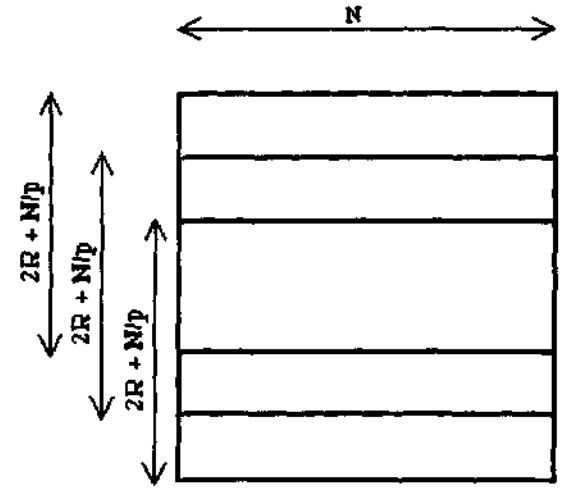

Fig. 3. The input image is divided into p overlapping subimages, and a processor is assigned to each one. Here $p = 3$.

4.1. Concurrency in symmetry detection

A study of the sequential algorithm presented in the previous section reveals the data parallelism inherent in the symmetry detection problem. Recall that the symmetry detection algorithm employs filters h_n of size $(2R+1) \times (2R+1)$ to compute one pixel $O(i,j)$ of the output image, where R is the chosen radius. The time required to process a single window is $O(R^2)$. The number of windows to be processed is the same as the number of output pixels, namely $(N-2R) \times (M-2R)$ for an $N \times M$ input image. Thus for a square image of length N, the sequential time complexity of symmetry detection is $O(R^2N^2)$. We note that all the output pixels can be computed independently of one another by concurrently processing all the windows. If a massively parallel machine with $O(N^2)$ processors were available, it is possible to conceive a parallel machine that would execute the symmetry detection algorithm in $O(R^2)$ time.

In our work, we use a more modest form of multiprocessing, where a constant number of processors, say p, is available. We divide the output image into p nonoverlapping strips of size $(N-2R)/p \times (N-2R)$. A parent task, which runs on the host nodes where the

symmetry detection computation is initiated, spawns p children tasks which run on other computational nodes that comprise the distributed computing environment. PVM allows the user to dynamically attach nodes to or detach nodes from the distributed computing environment. The mapping of tasks to computational nodes is the responsibility of the Parallel Virtual Machine (see Section 4.2). In our application, each of the p tasks is assigned to one of the output image strips. Thus the number of pixels processed by each processor is $(N-2R)(N-2R)/p$. We can expect a speedup of p since there is little overhead of parallel processing. It should be noted that this estimate of speedup is not severely affected even if the communication overhead were taken into consideration since the computational complexity of the parallel algorithm is higher than its communication complexity.

We devised an image preprocessing algorithm which divides the input image into nearly equal sized subimages and stores them in separate files on the network file system (NFS). Thus, given p computational nodes, our algorithm divides the $N \times N$ input image into overlapping subimages of size $(2R + N/p) \times N$. Fig. 3 illustrates this scheme. The subimages are stored as separate files with integer filename extensions which stand for task identification numbers. The subimages are read by the respective tasks, which compute the relevant output pixels and store back the output image on the network file system. Again, task identification numbers are used as filename extensions so as to avoid filename conflicts. The parent task is responsible for merging these output subimages into the final output.

4.2. PVM (Parallel Virtual Machine)

PVM (Parallel Virtual Machine) [15,14] provides a software environment for distributed computing on a network of heterogeneous computing nodes. Thus the computing nodes could be personal computers, engineering workstations, or supercomputers. A single application can exploit the variety of computational resources available on the network through the programming environment provided by PVM. We employed PVM 3.1 over a local area network (ethernet) consisting of both Sun/3© and Sun SPARC10© workstations in our implementation. Under the PVM environment, all the computational nodes on the network appear like a single logical distributed-memory multiprocessor. PVM permits the user to create tasks (units of computation) on the logical machine. These tasks can communicate with one another using library functions provided by PVM. Although the individual computers on the network may have diverse architectures, PVM makes it possible for data communication among the tasks to be handled in a uniform manner by making data conversions transparent to the user. In order to use PVM, a user must start

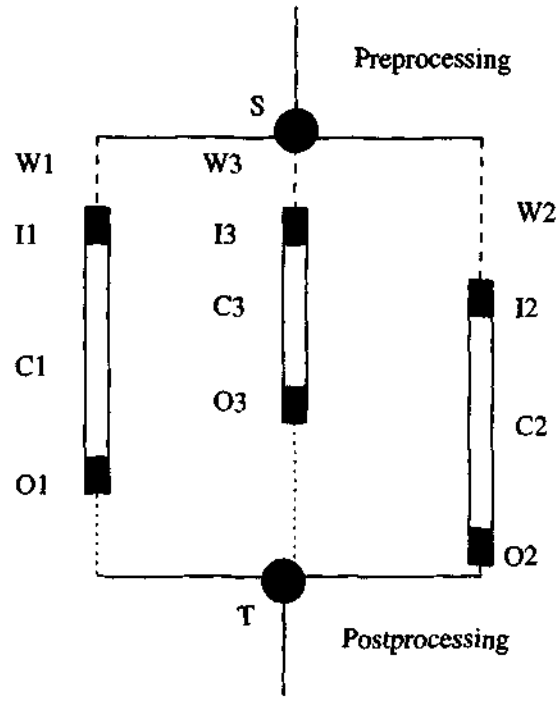

Fig. 4. Execution of a program in PVM environment.

the PVM daemon process on each node that the user wishes to include in the logical machine. A PVM program can be started as a UNIX command. The program can create children tasks through the library function pvm_spawn( ). An executable file must be named in conjunction with the function pvm_spawn( ), whose image will be created as a task on one of the nodes in the distributed computing environment. The user may name the machine on which the task must be created, or leave it to PVM to find a node on which the task will be started. Inter-task communication is carried out through the use of PVM library routines pvm_send( ), and pvm_recv( ). PVM also supports broadcast and multicast communications. If a node involved in distributed computation fails due to file I/O error or memory allocation error, the particular task is selectively killed by the parent task. In addition to the communication primitives supported by PVM, we have used the network file server (NFS) available on the network of Sun workstations to share data among the tasks. If two tasks wish to share a large amount of data, this sharing can be achieved through the use of files. Task i which must send the data to task j writes a file with filename extension j. Task i waits until the file with extension j is readable.

We employed static load balancing in our application. We estimated that the Sun SPARCstation 10 runs about five times faster than the Sun 3 under the type of load prevalent in our institution. Thus the images were divided in a non-uniform manner. Faster nodes receive about five times the load in comparison to the slower machines.

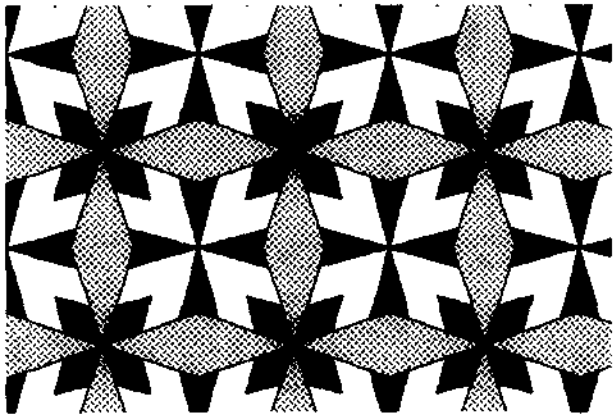

Fig. 5. Image for testing symmetry detection algorithms, showing both rotational and reflectional symmetry.

Fig. 4 shows the execution of a program in the PVM environment. S represents the time instance at which children tasks are spawned by the parent task. T represents the synchronization point i.e. the time instance at which all the children tasks complete their processing and report to the parent task. In the figure, we have illustrated the execution of three concurrent tasks. The life time of each task consists of the following phases: a waiting phase W_i during which the task has either not yet been assigned to any computing node or is waiting for CPU-time on the node to which the task has been assigned. The next phase of a task is the input phase I_i, where the subimage is read from an input file. This phase is followed by the computational phase C_i where symmetry tests are applied and pixels corresponding to the output subimage are computed. What follows next is the output phase O_i during which the output subimages are written to an output file. After phase O_i, the task i reports to the parent task. The parent task employs a barrier synchronization to ensure that all the children tasks finish before the parent can merge the output subimages into a single file. The execution time of the parallel algorithm can be written as the sum of the image

Fig. 6. Result of reflection symmetry detection shown superimposed on the original image. The white pixels are those where the reflection index (14) exceeds the threshold $T = 0.95$.

Fig. 7. Result of four-fold rotation symmetry detection shown superimposed on the original image. The white pixels are those where the rotation index (12) exceeds the threshold $T = 0.95$.

preprocessing time, the time for (parallel) computation of the output image, and the postprocessing time. As mentioned earlier, image preprocessing consists of splitting the original image into subimages, and image postprocessing consists of merging the output subimages. A call to the procedure *get_rusage*( ) (get resource usage) before the preprocessing begins on the parent task and a second call to *get_rusage*( ) after the postprocessing phase are used to estimate the CPU-time expended in executing the preprocessing and postprocessing phases on the parent task. From Fig. 4, it is clear that *get_rusage*( ) UNIX system call cannot be used to estimate the time that elapses from instance S to instance T. We used the difference in wall clock time between time instances T and S to estimate the time for parallel computation of the output image.

4.3. Results and discussion

The symmetry detection algorithms of Section 3 were first implemented as a program in Matlab© software and tested on an IBM PC/486 with 16 MB memory. We observed that for images of a reasonable size, such as

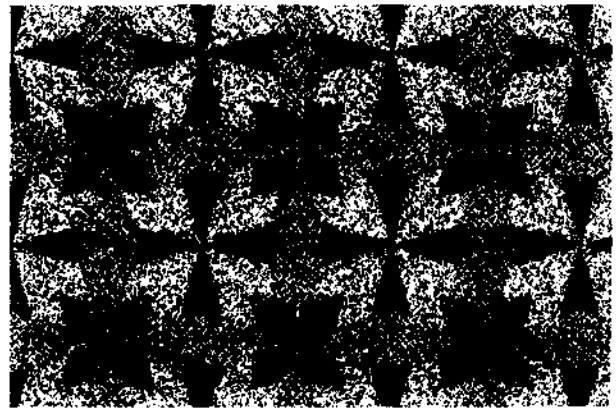

Fig. 8. Noisy image to test robustness of symmetry detection algorithms.

Fig. 9. Result of reflection symmetry detection test on the noisy image (shown superimposed). The same threshold $T = 0.95$ as used in the previous figures has been applied.

256×256 pixels, the symmetry tests are very slow, taking 30–50 minutes of computation time per image. This motivated us to look for faster algorithms; as a first step, we implemented the same algorithms on a Sun/SPARC workstation using the C programming language. This serial version of the C program ran over 10 times faster than the PC-Matlab version. However, the performance was still poor, and we were motivated to develop the parallel algorithm of Section 4, which ran substantially faster.

The parallel algorithm was tested on images of different sizes, all generated from the basic image shown in Fig. 5. This basic image is useful for testing because it displays considerable amount of reflectional as well as rotational symmetry. The results of the reflectional and rotational detection tests applied to the image in Fig. 4 are shown in Fig. 6 and Fig. 7 respectively. The results show that the proposed symmetry detection algorithms correctly detect the presence of symmetry. The algorithms also mark numerous points that do not appear, at first glance, to be true symmetry locations. However on closer inspection, it is seen that the actual image data is symmetric, but our perception of symmetry

Fig. 10. Result of four-fold rotational symmetry detection test on the noisy image (shown superimposed). The same threshold $T = 0.95$ as used in the previous figures has been applied.

Table 1
Speedup of reflectional symmetry test. Image size is 210×317, with $R = 20$

No. of tasks	Execution time (sec)	Speedup
1	1618.02	1.00
2	760.12	2.13
4	398.90	4.06

is affected by many factors that are not well modeled by the numerical tests (14) and (16). To verify the robustness of the symmetry detection algorithms, the same symmetry tests were applied to the noisy image shown in Fig. 8. The results generated for the noisy image are shown in Figs. 9 and 10. The reader may notice that the number of "false positives" is higher with the noisy images, but the true locations of symmetry are accurately detected.

The execution times of our programs are given in Tables 1, 2, and 3 for various image sizes and mask radii when used with the reflectional and rotational symmetry tests. The time required for preprocessing and postprocessing has been taken into account in calculating the speedups reported in these tables. Thus the overhead of parallel processing has been considered in our performance evaluation. Table 3 shows an instance of superlinear speedup when the number of tasks is four. This may be explained as follows. The workstations are, in general, heavily loaded. Because of this reason and due to the timesharing environment, the computationally intensive serial versions of the symmetry programs run with low priority. In the parallel implementation, since the image is partitioned into several parts, the memory requirement and computational requirement of each task is smaller. As a result, not only do the processes run with higher priority, they lead to better performance due to better memory management (cache as well as main memory). Thus the overall work done by the parallel implementation may well be less than the total work done by the sequential implementation.

The computational platform available at IIT Delhi, where this work was carried out, consists of two Sun/Sparc 10© and eight Sun/3© machines, the latter being slower than the former. We studied the effect of such a

Table 2
Another test of speedup of reflectional symmetry test by parallel implementation. Here, the image size is 128×128, with $R = 2$

No. of tasks	Execution time (sec)	Speedup
1	10.417	1.00
2	5.200	2.00
4	2.623	3.97
8	1.316	7.92
16	0.663	15.47
32	0.335	31.06

Table 3
Speedup of fourfold rotational symmetry detection by parallel implementation. Image size is 128 × 128, with $R = 16$

No. of tasks	Execution time (sec)	Speedup
1	153.64	1.00
2	79.72	1.92
4	40.11	3.83

heterogeneous computing platform on our parallel algorithm. When fewer numbers of processors were included for the parallel processing job, the time taken was considerably larger depending on whether the slower machines were included. This is due to the fact that the final output could be obtained only after all the subtasks were completed and considerable time was spent on waiting for the slower machines to finish processing.

It was also observed that the performance of the parallel algorithm depends significantly on how much load is present in the communication network. The non-uniform scheduling of the tasks, which results from the large load on the network, results in sublinear speedup as in the case of the rotational symmetry detection.

The results of the parallel and sequential versions of symmetry detection algorithms were compared for probable changes in the quality of the output. From the outputs, we observed that the output of the parallel algorithm is not disturbed in any way, i.e., the results of the parallel algorithm are identical to those of the sequential algorithm.

The concurrency present at the pixel level can be exploited to solve the problem of symmetry detection if many processors are available. This would require a change in the approach we have chosen here. Since the number of processors available were limited, our algorithm limits the number of subtasks that can be started for a given input image and the selected search radius. Alternative strategies for massively parallel machines form an interesting topic for future research.

5. Conclusions

Image processing and computer vision are application areas that can benefit from parallel processing. In this paper, we have considered the detection of reflectional and rotational symmetries in plane images. We have developed and implemented a parallel algorithm on a network of workstations. Our algorithm was tested to detect reflectional and rotational symmetry in images of different sizes. Our experimental results indicate significant speedup in relation to sequential processing while maintaining the quality of output. It is possible to further improve these results by exploiting the presence of concurrency at the pixel level and using a large number of processors.

Acknowledgements

We thank the two anonymous referees for their suggestions to improve the paper. We thank the Computer Services Center of IIT Delhi for permitting us to use the computational resources which were required to carry out this work.

References

[1] T. Zielke, M. Brauckmann and W. Von Seelen, Intensity- and edge-based symmetry detection with application to car following, Computer vision, graphics, and image processing: Image understanding, 58 (1993) 177–190.
[2] V. Govindaraju, S.N. Srihari and D.B. Sher, A computational model for face location, in Proc. Third Int. Conf. on Computer Vision, Osaka, Japan, December 1990, pp. 718–721.
[3] J. Bigün and J.M.H. du Buf, N-folded symmetries by complex moments in Gabor space and their application to unsupervised texture segmentation, IEEE Transactions on Pattern Analysis and Machine Intelligence, 16 (1994) pp. 80–87.
[4] A. Kuehnle, Symmetry-based recognition of vehicle rears, Pattern Recognition Letters, 12 (1991) pp. 249–258.
[5] R. Kakarala and J. Sivaswamy, Determining the symmetries of patterns in images, in Proc. 1994 European Conf. on Artificial Intelligence: Workshop on Spatial and Temporal Reasoning, Amsterdam, August 1994.
[6] M.A. Armstrong, Groups and Symmetry, Springer-Verlag, Berlin, 1988.
[7] J.G. Proakis and D.G. Manolakis, Digital Signal Processing: Principles, Algorithms, and Applications, Maxwell-McMillan, New York 2nd edn., 1992.
[8] B. Grunbaum and G.C. Shephard, Tilings and Patterns, Freeman, New York, 1989.
[9] A. Papoulis, Probability, Random Variables, and Stochastic Processes. McGraw-Hill, New York; 3rd edn., 1991.
[10] V.K. Prasanna, (ed.), Parallel Architectures and Algorithms for Image Understanding, Academic Press, New York, 1991.
[11] C.P. Ravikumar and R. Sethi, SHARP: A shape recognition system and its parallel implementation, Microprocessors and Microsystems, 19 (1995) 131–138.
[12] R. Nevatia, (ed.), Heterogeneous computing for vision, in Proc. of the Heterogeneous Computing Workshop, Cancun, Mexico, April 1994.
[13] C.-K. Lee and M. Hamdi, Parallel image processing on a network of workstations, Parallel Computing, 21 (1995) 137–160.
[14] A.L. Beguelin, PVM 3.0 – Parallel virtual machine 3.0, Technical report, Department of Computer Science, University of Tennessee, Knoxville, TN, 1992.
[15] A. Geist et al., PVM 3.0 User's Guide and Reference Manual, Engineering Physics and Mathematics Division, Oakridge National Laboratories, Tennessee, Report ORNL/TM-12187, May, 1993.
[16] K. Hwang, Advanced Computer Architecture: Parallelism, Scalability, Programmability, McGraw-Hill, New York, 1994.

R. Parthiban was born in Tamil Nadu, India in 1965. He obtained an M.Sc degree in Mathematics from the University of Tamil Nadu in 1987. He obtained a Master of Technology degree in Computer Applications from the Department of Mathematics in Indian Institute of Technology in 1995. He is presently employed as a Software Engineer at Pentech Computers Ltd., India.

C.P. Ravikumar received a Bachelor's degree (Electronics) from Bangalore University, India in 1983 and Master's degree (Computer Science) 1987. He obtained a Ph.D (Computer Engineering) from the University of Southern California in 1991. He joined the Department of Electrical Engineering, Indian Institute of Technology, New Delhi, in 1991. He is currently an Associate Professor in Electrical Engineering. His research interests are in the areas of VLSI design and Parallel Processing. He is the author of the book Parallel Methods for VLSI Layout published by Ablex Publishers, New Jersey (1996). Ravikumar Serves as the Indian Editor of the International Journal of VLSI Design (Gordon & Breech).

Ramakrishna Kakarala received the B.S. in Computer Engineering, and the M.S. in Electrical Engineering from the University of Michigan, Ann Arbor, in 1986 and 1988, respectively. During the summers of 1985–1987, he was with the Perception Laboratory, Naval Ocean Systems Center–Hawaii. He received the Ph.D. degree in Mathematics from the University of California, Irvine, in 1992. While at Irvine, he held a Chancellor's Fellowship. He has been with University of Auckland as a Lecturer in the Department of Electrical and Electronic Engineering since 1992. His current research interests include parameter estimation techniques and performance bounds, and their applications in computer vision.

Jayanthi Sivaswamy received a B.S. degree in Electrical Engineering, in 1984 from the Rochester Institute of Technology, Rochester and M.S. and Ph.D. degrees in Electrical Engineering from Syracuse University, Syracuse, in 1986 and 1992 respectively. After a brief period in the Indian Institute of Technology-Delhi, she joined The University of Auckland, in 1993 where she currently is a lecturer in the department of Electrical & Electronic Engineering. Her current research interests include natural vision based computer vision systems and modelling biological vision.