Parallel Implementation of Vision Algorithms on Workstation Clusters[*]

Dan Judd, Nalini K. Ratha, Philip K. McKinley, John Weng, and Anil K. Jain

Department of Computer Science
Michigan State University
East Lansing, Michigan 48824
{danjudd, ratha, mckinley, weng, jain}@cps.msu.edu

Abstract

Parallel implementations of two computer vision algorithms on distributed cluster platforms are described. The first algorithm is a square-error data clustering method whose parallel implementation is based on the well-known sequential CLUSTER program. The second algorithm is a motion parameter estimation algorithm used to determine correspondence between two images taken of the same scene. Both algorithms have been implemented and tested on cluster platforms using the PVM package. Performance measurements demonstrate that it is possible to attain good performance in terms of execution time and speedup for large-scale problems, provided that adequate memory, swap space, and I/O capacity are available at each node.

1 Introduction

Many tasks in computer vision are computationally intensive. In order to reduce the total time required to solve these problems, special-purpose hardware or vector supercomputers have often been used. Recently, parallel processing has started to dominate the supercomputer industry. One trend in parallel computing is towards the use of *clusters*, in which collections of off-the-shelf computers are connected to form a parallel computer. A *workstation cluster* comprises general-purpose workstations interconnected by a local area network (LAN). In a *cluster-based supercomputer*, such as the IBM SP1, the system is designed as a parallel computer from the outset, often using high-speed networks to interconnect nodes. Besides their cost advantage over other architectures, clusters are characterized by very powerful nodes with large memories and I/O capacities. Also, clusters are relatively flexible, in that additional computing and communication capacity, can be easily configured into the system.

In this paper, we report the results of implementing and testing two computer vision algorithms on cluster platforms. The first is a square-error clustering algorithm, which is used for image segmentation [1] in such applications as document image processing [2]. The second is an image matching algorithm [3], which is used to determine correspondence between two images taken of the same scene.

The goal of this research is threefold: First, we seek to understand how to implement these particular algorithms so that they make effective use of the cluster platforms and achieve good performance. Second, we seek efficient solutions to the communication and scheduling problems that arise when executing such numerical scientific applications in distributed environments. Third, we want to develop programs that can be easily installed and used by other image processing researchers, most of whom have access to at least one cluster environment.

The remainder of the paper is organized as follows. Section 2 presents necessary background information. Sections 3 and 4, respectively, present the data clustering and image matching subprojects. Each section describes the sequential algorithm, the parallel implementation, and experimental results. Section 5 concludes the paper.

2 Background

This section briefly describes the two computer vision algorithms that were studied, followed by a description of the distributed programming method used.

Data Clustering. One of the most fundamental problems in image processing is the need to group pixels with similar characteristics into distinct clusters and, thus, allow regions of interest to be targeted for further processing. The two key elements for good

[*]This work was supported in part by the NSF grants CDA-8806599, MIP-9204066, CDA-9121641, CDA-9222901, by DOE grant DE-FG02-93ER25167, by an Ameritech Faculty Fellowship, and by a grant from the Northrop Corporation.

clustering of any data set are the selection of the features that allow for good separation of "homogeneous" regions and the method for deciding, based on these features, to which region a given data point should belong. The problems of feature definition and selection are beyond the scope of this paper.

The general clustering method that we use is a variant of "CLUSTER" [4], which uses a square-error clustering approach [5]. The algorithm, which was first developed in the late 1960s [6], is computationally-intensive, and the number of operations grows linearly with the number of clusters being sought. The calculations are relatively independent, however, and therefore the algorithm lends itself to parallelization. In the project described herein, the CLUSTER program was redesigned to execute in parallel. The resulting program, termed P-CLUSTER, was implemented and tested atop several cluster platforms.

Image Matching. Image matching refers to the process of determining correspondence between two images so that the matched points in the two images correspond to the same physical point in the scene. Vision problems that require the matching include: structure from stereo, structure from motion, and motion estimation. For motion analysis, establishing correspondence between different perspective views of the same scene is critical. The existing techniques for this purpose can be categorized as either continuous or discrete. In the continuous approach, the velocity field is computed at each point in the image. Usually, there is no explicit feature extraction and matching. In the discrete approach, a set of features is computed and matched. Features proposed in the literature include points of high intensity variations, closed contours of zero crossings, and edges. Methods in both the categories have their advantages and disadvantages.

Weng *et al.* [3] proposed a new image matching method that combines the advantages of both the discrete and continuous methods. The algorithm for image matching using this technique, discussed later, has been parallelized and implemented on a cluster of Sun Sparc-10 workstations connected by Ethernet.

Programming Method. The package we used to implement both distributed algorithms is a TCP/IP-based communication library called PVM (Parallel Virtual Machine) [7]. PVM is public domain software from Oak Ridge National Laboratory, which provides an infrastructure for network-based heterogeneous concurrent computing. While we also tested other packages, the performance of the resulting programs was not significantly different than that of the PVM implementation.

3 Data Clustering Study

Square-error clustering seeks a partition of the data that, for a fixed number of clusters, minimizes the square-error. Suppose that a set of n patterns in d dimensions (also called *features*) is partitioned into K clusters, $\{C_1, C_2, \ldots, C_K\}$, with cluster C_k containing n_k patterns and each pattern assigned to a unique cluster. The *centroid* of cluster C_k is defined as $\mathbf{m}^{(k)} = (1/n_k)\sum_{i=1}^{n_k} \mathbf{x}_i^{(k)}$, where $\mathbf{x}_i^{(k)}$ is the i^{th} pattern belonging to cluster C_k. The square-error e_k^2 for each cluster C_k is the sum of the squared Euclidean distances between each pattern in C_k and its centroid, and the square-error for the entire K-cluster partition is $E_K^2 = \sum_{k=1}^{K} e_k^2$.

Many algorithms for square-error clustering use an iterative process, in which an initial partition is selected and patterns are reassigned to clusters so as to reduce square-error, with the process repeated until the cluster membership stabilizes. The process of assigning all patterns to the cluster with the closest centroid is referred to as a *K-means pass* [8]. Some algorithms use a *forcing pass*, which perturbs the current partition before the final assignment of patterns, in an effort to avoid convergence to a local minimum.

Sequential Algorithm. The CLUSTER program makes use of both a K-means pass and a forcing pass. CLUSTER attempts to find the "best" partition containing $1, 2, \ldots, KMAX$ clusters, for some limiting value $KMAX$ specified by the user. First, a K-means pass is used to create a sequence of clusterings containing $2, 3, \ldots, KMAX$ clusterings, where the starting point for a K-cluster solution is based on the result of a $(K-1) - clustersolution$. Next, the forcing pass creates another set of clusterings by merging existing clusterings, two at a time, to see if a better clustering can be achieved. Due to space limitations, details of the algorithm are omitted here, but may be found in [9].

Distributed Implementation. We implemented the distributed algorithm with a client-server model and coarse-grained parallelism. Data is partitioned into *blocks*, and each block is assigned to a *client* process, which knows the values of all the (current) centroids. The client computes the distances for each pattern in its block and assigns each pattern to the appropriate cluster. The client also calculates the *block partial sum* for each cluster, over the patterns in its block, and sends the results to a *server* process. Specifically, for each cluster C_k, the client responsible for block b computes $P_{k,b} = \sum_{i=1}^{n_{k,b}} \mathbf{x}_i^{(k,b)}$, where $n_{k,b}$ is the number of patterns in block b that are assigned to cluster C_k, and $\mathbf{x}_i^{(k,b)}$ is the i^{th} pattern vector in block b belonging to cluster C_k. When the server has collected block partial sums from all the clients, it calculates the new centroids and returns them to the client processes, which begin a new iteration.

The original CLUSTER program used McQueen's method [8] to calculate new centroids. This method is inherently sequential, however, and also was found to lead to problems with roundoff error on large data sets. To avoid these problems, our implementation only updates the centroids *after* all patterns have been assigned to a cluster, which is the same method that was used originally by Forgy [10].

Experimental Results. We applied P-CLUSTER to standard texture images [11], each of which had been passed through a set of Gabor filters to produce 20 features per pixel. Jain and Farrokhnia [1] previously used the sequential CLUSTER program to segment a variety of such textured images, however, because of the computational requirements of CLUSTER, they used only a small number (approximately 6 percent) of randomly chosen pixels in the image as input to the clustering algorithm.

Figure 1 illustrates the application of the P-CLUSTER program to a 512×512 image. Figure 1(b) shows the results of executing P-CLUSTER on the image in Figure 1(a). Although Jain and Farrokhnia [1] were able to achieve somewhat better clustering results on this particular problem by using additional heuristics, the emphasis in this paper is on the parallelization of the clustering algorithm.

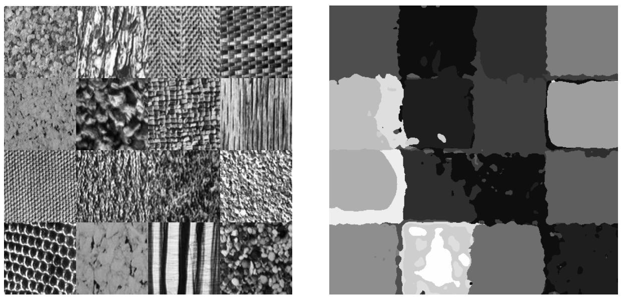

(a) original image (b) P-CLUSTER result

Figure 1. P-CLUSTER performance on a 512×512 image with 20 features.

The P-CLUSTER program was tested atop several cluster platforms. Due to space constraints, only a sampling of the results can be presented here; additional performance results can be found in [9]. Table 1 gives the timing results of the CLUSTER and P-CLUSTER programs on the 512×512 image shown in Figure 1, as executed atop a cluster of Sun Sparc-10 workstations interconnected by Ethernet. Each node has 32 Mbytes of main memory. The *core* of the P-CLUSTER program is defined to be the part of the program where parallelization occurs. The execution of P-CLUSTER on multiple workstations achieves greater than an n-fold improvement with n processors due to excessive paging on one workstation. The aggregate memory of several workstations can easily accommodate large space requirements, thereby avoiding thrashing.

We also tested the P-CLUSTER program on a 128-node IBM SP1 located at Argonne National Laboratory. Each node is equipped with 128 Mbytes of main memory and a 1 Gbyte disk. Table 2 gives the timing results of execution of the CLUSTER and P-CLUSTER programs on the SP1 for the same 512×512 image. Here we see the dramatic effect on the sequential time that results from having sufficient main memory on each node. On the Sparc-10, the sequential P-CLUSTER program required 51.1 hours, whereas on a single node of the SP1, the same problem required only 2.4 hours. Some superlinear speedups still appear, due to improved cache performance with smaller data sets on each node.

Table 1. P-CLUSTER: Sparc-10s, 512×512 image.

Program and Environment	Metric	Execution Time	Parallel Speedup
CLUSTER (sequential Fortran)	total time	55.5 hrs	—
P-CLUSTER (sequential C)	core time	50.9 hrs	1.0
	total time	51.1 hrs	1.0
P-CLUSTER (4 processors)	core time	112.5 min	27.1
	total time	137.0 min	22.3
P-CLUSTER (8 processors)	core time	38.6 min	79.1
	total time	52.7 min	58.1
P-CLUSTER (16 processors)	core time	22.7 min	134.1
	total time	36.6 min	83.7

Table 2. P-CLUSTER: SP1, 512×512 image.

Program and Environment	Metric	Execution Time	Parallel Speedup
CLUSTER (sequential Fortran)	total time	—	—
P-CLUSTER (sequential C)	core time	8289 sec	1.0
	total time	8578 sec	1.0
P-CLUSTER (4 processors)	core time	2169 sec	3.8
	total time	2725 sec	3.1
P-CLUSTER (8 processors)	core time	891 sec	9.3
	total time	1187 sec	7.2
P-CLUSTER (16 processors)	core time	582 sec	14.2
	total time	863 sec	9.9

Figure 2 plots the execution times for the Sparc-10 cluster and the SP1. The workstation cluster performs well, considering that each node (Sparc-10 model 30) is equipped with less memory and a slower processor (36 MHz compared to 62.5 MHz).

4 Image Matching Study

The image matching algorithm we used operates on two different images of the same scene. The algorithm is designed to deal with large image disparities and provide dense image matching (a matching vector for every pixel) [3].

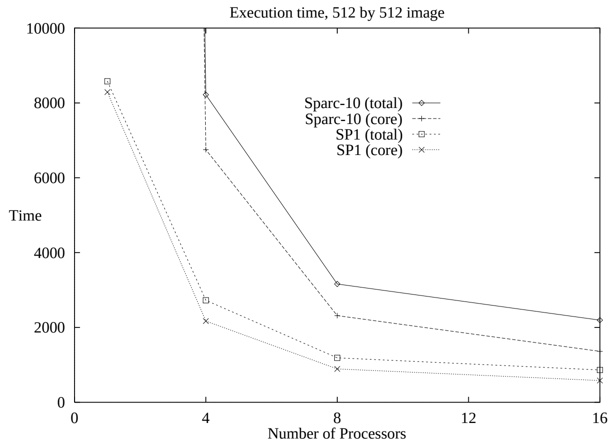


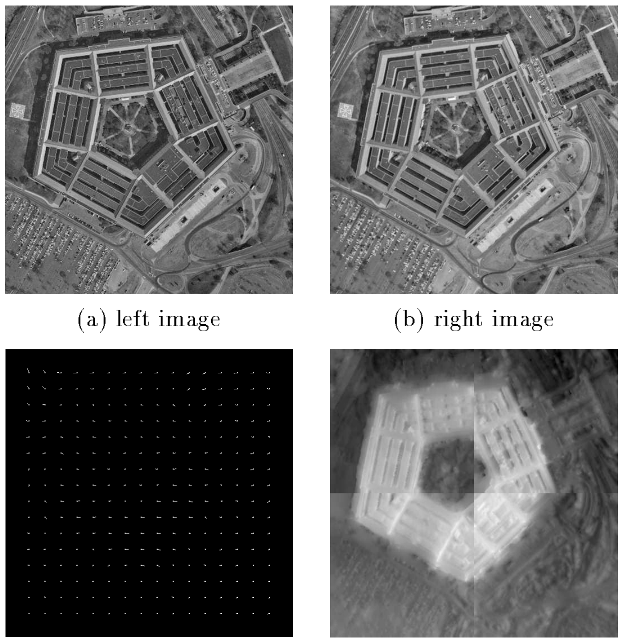
Figure 2. Execution times (seconds) for P-CLUSTER on a 512 × 512 image.

Sequential Algorithm. When implemented sequentially, the algorithm comprises the following two major steps:

1. Compute the four attribute images: edgeness, positive cornerness, negative cornerness, at each pixel in both the images.
2. For level l from 7 down to 0, perform the following steps:
   (a) Blur the attribute images to level l; the spatial extent (scale) of blurring at level l is double of that at level $l-1$.
   (b) Compute the displacement field at each grid point by evaluating an equation [3]. Each such evaluation constitutes a single iteration in a series of iterations that progressively minimize the weighted sum of squares of residuals in intensity, positive and negative corners, edgeness, orientation smoothness, and the displacement smoothness. Perform a fixed number of iterations (say 20 as recommended in [3]). The density of the grid points at level l is one-half of that at level $l-1$.
   (c) If l is not equal to 0, project the displacement field computed at level l to the grid at level $l-1$, by copying the vector at each grid point to the four corresponding grid points of level $(l-1)$.

Distributed Implementation. As with clustering, we adopted a master-slave model in the parallel implementation of the matching algorithm. The workstation where the job starts is considered as the master. A spatial data partitioning approach was used to generate individual tasks, with each smaller-sized image segment sent by the master to a physically separate slave node. Each slave carries out the task assigned to it and, upon completion, communicates the results to the master, which carries out the final result integration. In order to simplify the image partitioning and without any loss of generality, we have assumed a square number of workstations in the cluster.

Experimental Results. The matching algorithm was tested using images of size 512×512. The two input images (left and right), the resulting (u-v) displacement map, and the depth map are shown in Figure 3.



(a) left image (b) right image

(c) displacement (uv) map (d) depth values

Figure 3. Matching results on Pentagon image.

First, the sequential algorithm was executed on a single workstation. In order to understand how execution time was distributed among the submodules of the algorithm, the standard tool *prof* was used to produce an execution profile of a program. Table 3 gives the time spent in various stages of the algorithm for a 512×512 image with 7 levels of pyramid, as well as the total execution time, on both a Sun Sparcstation-2 and Sparcstation-10 workstation.

The parallel version of the algorithm was implemented on a workstation cluster connected on a LAN with file servers. All the workstations were Sun Sparcstation-10's.

We implemented two versions of the distributed algorithm, both based on PVM. In the first approach, writing of intermediate results to disk is handled through the Network File Server (NFS). In order to reduce network traffic, the second approach allows each process to write the intermediate results to the local disk of its workstation. Finally, we needed a measure to evaluate the distributed implementation against the

Table 3. Execution profile (in seconds) of sequential image matching algorithm on a 512×512 image

Activity	Sparc-10 time	Sparc-2 time
Attribute computations	18.1	33.7
File creation and level 6	62.8	114.1
Level 5	55.6	96.6
Level 4	46.3	85.4
Level 3	44.8	89.4
Level 2	67.0	167.3
Level 1	189.3	534.5
Level 0	735.6	2019.2
Other	20.5	119.9
Total	1240.0	3268.1

sequential version. The following error measure E defines the average error in the displacement maps. The smaller the value of E, better is the result. Let N equal number of pixels in a row (or column), let u_1, v_1 be the u-v map of distributed version of the algorithm, and let u_2, v_2 be the u-v map of original (sequential version) algorithm. Then we have:

$$E = \frac{\sqrt{\frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}(u_1[i,j]-u_2[i,j])^2+(v_1[i,j]-v_2[i,j])^2}}{\sqrt{[max(u_1)-min(u_1)]^2+[max(v_1)-min(v_1)]^2}}.$$

Table 4 summarizes the total execution times for the two different implementations, as well as the error in the displacement maps, on the images shown in Figure 3. The speedup achieved using multiple workstations is plotted in Figure 4.

Table 4. Execution time (in seconds) and error for image matching.

No. of workstations	PVM/NFS	PVM with local disk storage	Error E
1	2034	2034	0.000
4	594	517	0.002
9	375	255	0.041
16	325	190	0.003
25	300	183	0.028

5 Conclusions

In this paper, we have presented results of a study of parallel algorithms for square-error data clustering and image matching, as implemented atop cluster platforms. Experiments showed that it is possible to attain good performance for both algorithms, provided that the nodes are equipped with adequate main memory, swap space, and I/O capacity. Moreover, in both cases the distributed implementations allow problem instances to be solved that are too large to be executed on a single workstation.

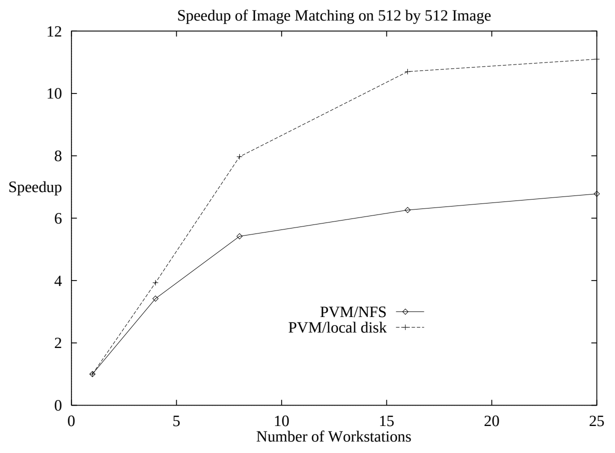


Figure 4. Comparison of speedup attained for two image matching methods.

References

1. A. K. Jain and F. Farrokhnia, "Unsupervised texture segmentation using gabor filters," *Pattern Recognition*, vol. 24, no. 12, pp. 1167–1186, 1991.

2. A. K. Jain and S. Bhattacharjee, "Text segmentation using gabor filters for automatic document processing," *Machine Vision Applications*, vol. 5, pp. 169–184, 1992.

3. J. Weng, N. Ahuja, and T. S. Huang, "Matching two perspective views," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, pp. 806–825, Aug. 1992.

4. R. Dubes and A. K. Jain, "Clustering techniques: The user's dilemma," *Pattern Recognition*, vol. 11, pp. 247–260, 1976.

5. A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data.* Prentice-Hall, 1988.

6. H. P. Friedman and J. Rubin, "On some invariant criteria for grouping data," *Journal of the American Statistical Association*, vol. 62, pp. 1159–1178, 1967.

7. V. S. Sunderam, "PVM: A framework for parallel distributed computing," *Concurrency: Practice and Experience*, vol. 2(4), pp. 315–339, Dec. 1990.

8. J. B. McQueen, "Some methods of classification and analysis of multivariate observations," in *Proceedings of Fifth Berkeley Symposium on Mathematical Statistics and Probability*, pp. 281–297, 1967.

9. D. Judd, P. K. McKinley, and A. K. Jain, "Cluster^2: Parallel data clustering on a workstation cluster," Tech. Rep. MSU-CPS-94-7, Department of Computer Science, Michigan State University, East Lansing, Michigan, Feb. 1994. submitted for publication.

10. E. Forgy, "Cluster analysis of multivariate data: Efficiency versus interpretability of classifications," *Biometrics*, vol. 21, p. 768, 1965.

11. P. Brodatz, *Textures: A Photographic Album for Artists and Designers.* New York: Dover, 1966.