Large-Scale Parallel Data Clustering

Dan Judd, Philip K. McKinley, *Member, IEEE*,
and Anil K. Jain, *Fellow, IEEE*

Abstract—Algorithmic enhancements are described that enable large computational reduction in mean square-error data clustering. These improvements are incorporated into a parallel data-clustering tool, P-CLUSTER, designed to execute on a network of workstations. Experiments involving the unsupervised segmentation of standard texture images were performed. For some data sets, a 96 percent reduction in computation was achieved.

Index Terms—Data clustering, mean square error, data mining, image segmentation, parallel algorithm, network of workstations.

1 Introduction

The amount of raw data available to researchers, in a variety of scientific fields, has been increasing at an exponential rate. Without automatic methods to process and manipulate such data, however, the value of accessing it may be overshadowed by the difficulty in its characterization [1]. A common method used in data exploration is *data clustering* [2]. Developing effective data clustering methods has been a longstanding problem in image processing. In document image understanding, for example, clustering can be used to separate regions of text and graphics [3]. In satellite image processing, clustering can be used in a variety of tasks, from classifying land usage to identifying strategic targets [4].

While many clustering methods have been applied successfully to image segmentation, mean square-error clustering [2], [5], has been one of the most popular methods, since it works well with little or no supervision by the researcher. Stated formally, a set of n patterns, or vectors, in d dimensions, is to be 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 ith 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 $E_K^2 = \sum_{k=1}^{K} e_k^2$.

The difficulty in square-error and other partitional clustering methodologies is that there is no computationally feasible method for guaranteeing that a given clustering minimizes the total square-error E_K^2. The number of possible assignments, even for small numbers of patterns and clusters, quickly becomes astronomical. Therefore, many clustering algorithms use iterative hill-climbing techniques that terminate

1) when no improvement can be made,
2) when the change in total error drops below a threshold, or
3) when a predetermined number of iterations has been completed.

• *The authors are with the Department of Computer Science and Engineering, Michigan State University, East Lansing, MI 48824.*
E-mail: {danjudd, mckinley, jain}@cps.msu.edu.

Manuscript received 19 May 1997; revised 29 Jun. 1998. Recommended for acceptance by G. Medioni.
For information on obtaining reprints of this article, please send e-mail to: tpami@computer.org, and reference IEEECS Log Number 107098.

However, even these algorithms are computationally intensive for large data sets.

Our research investigates the use of parallel processing to reduce the execution time of mean square-error data clustering. In the first phase of our study [6], we designed and implemented P-CLUSTER (for *Parallel*-CLUSTER), a parallel version of a square-error clustering algorithm. While a number of researchers previously have studied parallel data clustering on specialized architectures [7], [8], and on massively parallel processors [9], [10], our methods are designed to take advantage of the aggregate memory and computing power of a network of workstations (NOW). Experiments in segmenting standard texture images demonstrated that a small-scale NOW can effectively handle problems that overwhelm a single workstation. For example, clustering of a 512×512 texture image required over 50 hours to complete on a single Sun workstation, but only 70 minutes on four workstations, and 40 minutes on eight workstations [6].

In the second phase of our study, reported here, we have explored several variations on the mean square-error clustering algorithm itself. Our goal is to prune as much computation as possible while preserving the clustering quality. We accomplish this task through a combination of three algorithmic enhancements:

1) computing spheres of guaranteed assignment for cluster centroids,
2) computing the maximum movement effect for patterns across iterations, and
3) maintenance of partial sums for centroids.

In the remainder of this paper, we describe the basic P-CLUSTER algorithm, introduce the three major enhancements to the algorithm, and present the results of experiments on two NOW platforms: a collection of Sun workstations and an IBM SP-2 supercomputer. Due to space limitations, many details of our study are omitted here, but may be found in [11].

2 The Basic P-CLUSTER Algorithm

The square-error clustering algorithm used as a starting point for our study was originally developed in the late 1960's [5], and was later modified to create the CLUSTER program [2]. CLUSTER attempts to find the "best" clustering containing 1, 2, …, $KMAX$ clusters, for a specified value $KMAX$. The algorithm comprises two major components. First, a *K-means pass* [12] is used to create a sequence of clusterings containing 2, 3, …, $KMAX$ clusters, where the starting point for the kth clustering is based on the result of the $(k - 1)$st clustering. Next, a *forcing pass* [5] creates another set of clusters by merging existing clusters, two at a time, to determine whether a better clustering can be achieved.

The original CLUSTER program targeted small data sets (fewer than 4,000 patterns), and performed well in this domain. However, our experiments with CLUSTER showed that, when the problem size was larger than the available main memory on the system, disk thrashing occurred, and performance dropped sharply. Moreover, for large data sets, the round-off error introduced by the cluster updating method produced poor clustering results; in many cases, the algorithm simply failed to converge. These results made it clear that a major revision of CLUSTER was necessary. We chose a coarse-grained parallel structure suited to the NOW platform, and we modified the method for cluster assignment so as to reduce round-off errors.

The basic P-CLUSTER algorithm is given in Fig. 1. The algorithm uses a client-server process configuration. Data is partitioned into *blocks* by the server process, which sends the initial centroid list and work assignments (blocks) to each of the clients. For each block assigned to a particular client process, the client computes the distances for each pattern in the block, and assigns the pattern to the appropriate cluster. The client also calculates the

Algorithm: Parallel square-error clustering with forcing pass.
Input: *n* patterns with *d* features, maximum number of clusters, *KMAX*.
Output: History of each of the *KMAX* clusterings produced.

1) Server: Read from the user the value of the maximum number of clusters, *KMAX*. Read the pattern matrix and partition it into *n* blocks, where *n* is the number of clients involved.
2) Server: Distribute blocks of data to clients. Find the mean and standard deviation of the data. Perform data normalization if desired and set $K = 2$.
3) Server: For a given *K* clustering, an initial set of cluster centroids is chosen based on the centroids of the $(K - 1)$st clustering. The centroid of the cluster with the largest total variance is converted into two new centroids by adding (subtracting) the standard deviation of each feature of the cluster to (from) each pattern. The remaining new centroids are identical to the other $(K - 2)$ centroids from the $(K - 1)$st clustering.
4) Server: Broadcast the new centroids to all clients.
5) Clients: Each client assigns each pattern of its data block to the cluster whose centroid minimizes the Euclidean distance between the pattern and the centroid. For each cluster, the sum of all patterns assigned to that cluster is computed, on a feature-by-feature basis. The client sends to the server the number of patterns assigned to each cluster and the block partial sums for each cluster.
6) Server: Total the input from all the clients. If any of the clusters is empty, then recompute the initial centroids and go to step 4.
7) Server: If no pattern changed clusters or if a maximum number of iterations has been reached, then go to step 8. Otherwise, recompute each centroid based on the new clustering and go to step 4.
8) Server: If the number of clusterings found thus far is less than *KMAX*, return to step 3 in order to compute the $(K + 1)$st clustering.
9) Server: For a given *K* clustering, find the cluster pair that, when merged, causes the least increase in total error. Merge these two clusters and use the resulting centroid, along with the other $(K - 2)$ centroids, as centroids for a $(K - 1)$st clustering. Proceed as in steps 3 through 7 to compute clustering *K*. If the new clustering produces a lower total error than the previous one, then save the results.
10) Server: Repeat step 9 for $(KMAX - 1)$ clusters down to three clusters.
11) Server: Repeat steps 3 through 10 until no clustering is improved.

Fig. 1. P-CLUSTER algorithm.

block partial sum for each cluster, over the patterns in its blocks, and sends the results to the server. 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 clients, which begin a new iteration of assignments. When no further improvement is made during a given pass, each client sends the cluster membership of its data patterns to the server. The server then proceeds to the next pass or recognizes that termination conditions have been met and halts. In this algorithm, centroids are updated only after all patterns have been assigned to a cluster, which is the method originally used by Forgy [13]. This approach facilitates parallelization by enabling the block partial sums to be computed in parallel using any block size, independently of the other block partial sums. Moreover, computing the centroids after reassignment reduces round-off error because the original data is used to compute the partial sums in each iteration.

Our initial positive results with the P-CLUSTER program [6] encouraged us to explore methods to further reduce execution time. In our experiments, we observed that after a small number of passes through the data, relatively few patterns change their cluster assignments. Specifically, we noted that 70–80 percent of the changes in cluster assignment occur in the first two iterations, even though the clustering pass may require over 100 iterations to complete. As a result, the movement of centroids tends to decrease with each successive pass through the data. In this paper, we present three techniques that can be used to take advantage of these trends in order to reduce the number of distance calculations, and therefore, the total execution time, of mean square-error clustering. Other authors have taken advantage of this property in different types of clustering algorithms [14], [15]. In addition, we should point out that, while our methods are shown to be very effective in pruning computation, other proposed acceleration techniques, such as arranging centroids in a *K*-d tree [16], may complement these methods, and produce further performance improvement. Integration of such methods into P-CLUSTER is a subject of future research.

3 Computational Pruning Enhancements

3.1 Spheres of Guaranteed Assignment

The key to reducing computation time in square-error clustering is to minimize the number of distance calculations. One way to reduce distance calculations is to precompute a set of radii for each centroid, defining a set of *hyperspheres* that may preclude the need to check other centroids. Fig. 2 shows an example of this *Spheres of Guaranteed Assignment* method in two dimensions. Given a set of centroids, the distance between each pair of centroids is computed and saved in a table. In addition, stored with each pattern is its previously assigned centroid. If a pattern *P* is closer to a centroid C_i than half the distance to the centroid C_{min}, where C_{min} is the closest centroid to C_i, then *P* can be assigned to C_i without checking any other centroids. In Fig. 2, let us assume that C_2 is the closest centroid to C_1. Therefore, any pattern such as P_1, within a distance $\frac{|C_1 - C_2|}{2}$ of C_1 (depicted by the smallest solid circle around C_1), must be closer to C_1 than to C_2. Moreover, since any such pattern is necessarily closer to C_1 than to any other centroid C_i, only one distance calculation is needed to properly assign pattern P_1 to centroid C_1.

If the distance from a pattern *P* to C_1 is greater than half the distance from C_1 to C_2, then the distance to C_2 must also be computed. If the pattern is closer to C_2 than to C_1, then C_2 becomes the new minimum distance centroid, and the process is continued until no centroids remain to be checked or until *P* falls within a sphere of guaranteed assignment. Once again considering Fig. 2, let us assume that C_3 is the next closest centroid to C_1. Since pattern P_2 lies outside the first radius of C_1, the distance from P_2 to C_2 must be computed. When it is determined that P_2 is closer to C_1 than to C_2, the distance from P_2 to C_1 can be com-

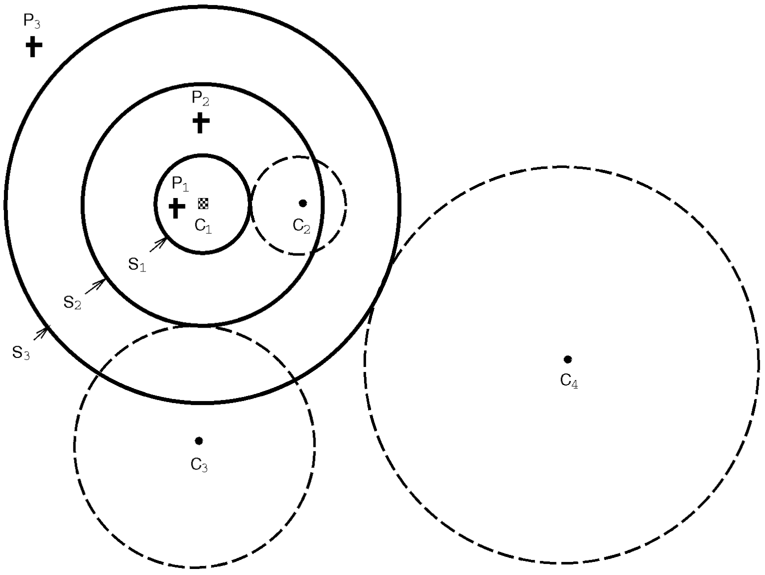


Fig. 2. Spheres of assignment example.

pared to the second sphere around C_1, corresponding to $\frac{|C_1 - C_3|}{2}$. Since the second radius is greater, P_2 need not be compared to any other centroid and can be assigned to C_1.

In Fig. 2, pattern P_3 lies outside all spheres of assignment, and thus the distance to every centroid must be computed in order to assign this point. It is obvious from casual observation that pattern P_3 is closer to centroid C_1 than to any other centroid, and it may be tempting to use a more inclusive shape, such as a hyperellipsoid or a convex hull of bounding planes, instead of a hypersphere. However, if the bounding shape itself becomes too complex, it may be computationally more expensive to determine whether a pattern is contained within the shape than simply to compute the distance from the pattern to every centroid. Clearly, the least number of centroid distance calculations occurs when the first centroid checked is the correct one. Storing the cluster to which each pattern was most recently assigned enables the algorithm to start with a likely candidate centroid.

3.2 Maximum Movement Effect

Given that the cluster membership of most patterns is unlikely to change in any given iteration, it is also advantageous to include a simple check to determine whether it is possible for the pattern to have changed membership at all. If the answer is no, then all computation for that pattern can be avoided and the assignment from the previous iteration can be used.

Formally, let P be a pattern and $\{C_1, \ldots, C_K\}$ be a set of K centroids, ordered in increasing distance from P, that is, $|P - C_1| \le |P - C_2| \le \cdots \le |P - C_K|$. We define the *maximum movement effect, M,* to be:

$$M = |P - C_2| - |P - C_1|.$$

M is the minimum total distance that C_1 and C_2 must move such that there is a possibility of P becoming closer to C_2 than to C_1. Clearly,

$$|P - C_i| - |P - C_1| \ge M, \quad \text{for all } i > 1,$$

since C_2 is at least as close to C_1 as is any other centroid, given the ordering.

Now, let all centroids $C_1, \ldots, C_K$ move to new positions $C_1', \ldots, C_K'$, and define m_i to be the distance moved by C_i:

$$m_i = \left|C_i - C_1'\right|, \ (1 \le i \le K).$$

Let m_{max} be the largest distance moved by any centroid other than C_1, specifically, $m_{max} = \max(m_i)$, $1 < i \le K$. The maximum distance that C_1 could have moved away from P is m_1, and the maximum distance that any other centroid C_i $(i \ne 1)$ could have moved closer to P is m_{max}. It can be shown that, if $M > (m_1 + m_{max})$, then P is as close to C_i' as to any other centroid C_i', $i > 1$ [11].

To make use of this result, stored with each pattern P is its M value, denoted M_P. As with the spheres of guaranteed assignment method, the previous cluster assignment is also stored with each pattern. (For the first iteration in each pass, M is set to -1 for all patterns.) Before each iteration, the distance moved by each centroid, along with m_{max}, is computed. For a pattern P that is a member of cluster i, P-CLUSTER sets $M_P = M_P - (m_i + m_{max})$. If $M_P \le 0$, then normal distance calculations must be executed to find both the closest centroid and the second closest centroid to pattern P. If $M_P > 0$, then all other computation on pattern P may be skipped. The new, reduced value of M_P is used in the next iteration. The reuse of M_P is possible because it is based on the total movement of all the centroids, which can take place over a series of iterations. The reduction of M_P by $(m_i + m_{max})$ is equivalent to maintaining a total sum of m_i and m_{max} for each pattern.

The method described above is based on a worst-case scenario in which the centroid associated with a pattern always moves directly away from that pattern, and all other centroids move directly toward the pattern. The actual P-CLUSTER implementation takes advantage of a minor optimization in order to improve performance; see [11].

3.3 Partial Sums

Experiments showed that the previous two methods substantially reduced the dominating computation of P-CLUSTER, namely the distance calculations, causing other components of the algorithm to become more prominent. One such component is the featurewise summation of all the pattern vectors within each cluster. Previously, the number of floating point additions required in this operation was small compared to the number of multiplications and additions involved in finding the closest centroid. As distance

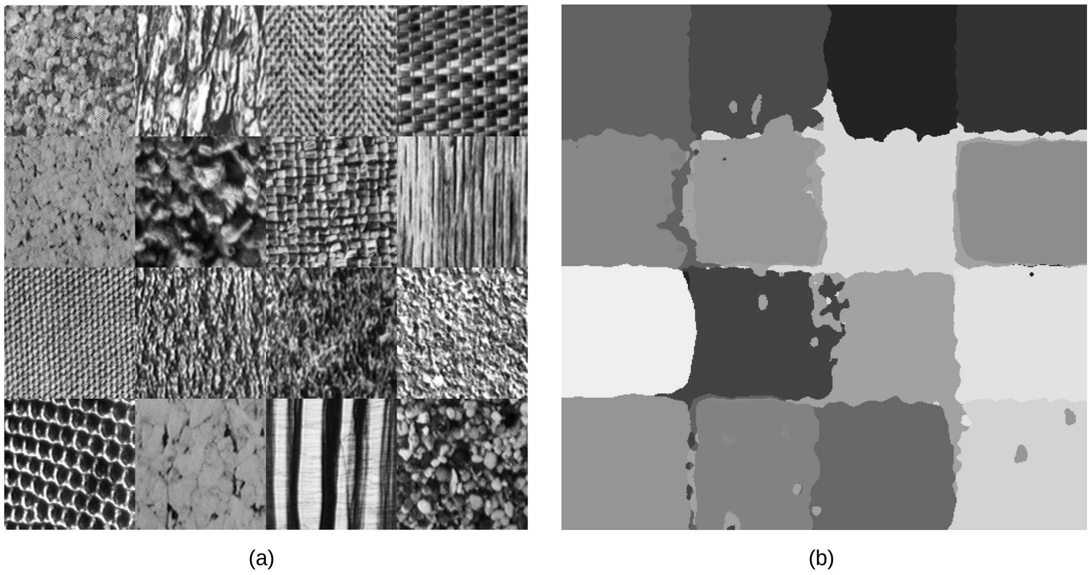


Fig. 3. P-CLUSTER performance on a 512 × 512 image with 20 features. (a) Input image. (b) P-CLUSTER result.

calculations were reduced, however, these addition operations gradually became the dominating computation, especially in later iterations when relatively few patterns change membership.

To reduce these computations, we maintained a set of partial sums associated with each block of data, one set for each possible cluster. Patterns that change assignment are subtracted from the old set of sums and added to the new set; a computation is carried out only when a membership change takes place. The potential problem with maintaining partial sums is the possibility that roundoff error will accumulate over time and eventually degrade the clustering results. To combat this phenomenon, we maintain a count of how many iterations have passed since the partial sums were created directly from the data. When a limit is reached, the partial sums are computed directly by summing the original data, thus limiting the potential effect of roundoff. We set the limit arbitrarily to 1,000, but in experiments with our two main data sets, no roundoff effects were observed, regardless of how high the limit was set. For other data sets and application domains, different values of this limit may be required.

4 Performance Evaluation

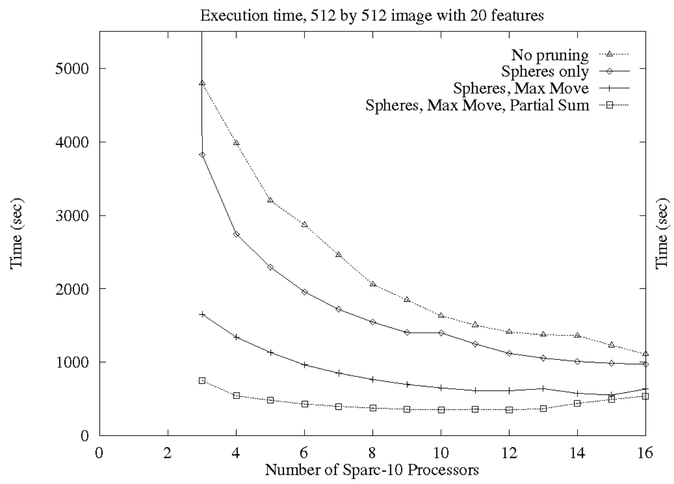
In order to evaluate the clustering quality and the execution time of the three enhancements to P-CLUSTER, we used the new versions of the program to segment standard texture images [17]. One such image, shown in Fig. 3a, is of size 512 × 512 pixels and contains 16 distinct textured regions. This image has been passed through a bank of 20 Gabor filters [18] to produce 20 energy features per pixel. The clustering result for this image is shown in Fig. 3b. We emphasize that all versions of the P-CLUSTER program produce identical clustering solutions for a given data set. Due to space limitations and for the sake of comparison, many of the performance results reported in this paper are for this particular image; results for other data sets can be found in [11].

We implemented the modified P-CLUSTER algorithm on a collection of workstations comprising a mixture of Sun Sparc-10 model 30's and model 40's with clock speeds of 33 MHz and 40 MHz, respectively, each with 32 Mbyte of memory and a relatively small (~200 Mbyte) local disk. The machines are interconnected via 10 Mbit/sec Ethernet. This configuration is typical of many laboratory environments. Interprocess communication was implemented initially using PVM (Parallel Virtual Machine) [19] and later using MPI (Message Passing Interface) [20]. We also ran a series of tests on an IBM SP-2 supercomputer at Argonne National Laboratory. Each node of the SP-2 is an IBM RS/6,000 model 370 workstation with a clock speed of 62.5 MHz, 128 Mbytes of memory, and a 1-gigabyte local disk. The nodes are interconnected by both a 10 Mbit/sec Ethernet network, and an 8.5 Mbyte/sec MIN-based high-speed switch.

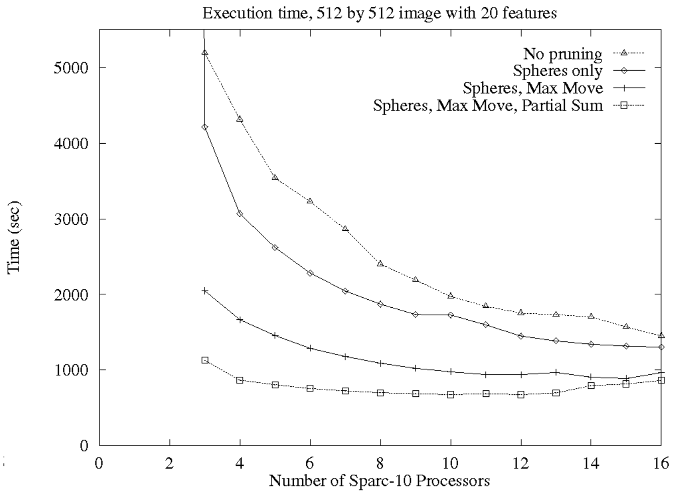
Two execution times are reported for each trial, a *total* time and a *core* time. The total time refers to the wall-clock time for a complete run of P-CLUSTER, including all data input and output. The core time refers to the time spent in the main computation section of the program. In general, the variance of core times was considerably less than that of the total times, due to the performance fluctuations of the network file system. In the following plots, the execution times when using one and two processors are sometimes excessive, due to thrashing, and are omitted. All data points are averages of multiple runs, usually five runs per point. Due to our limited availability of CPU time on the SP-2, on that platform five runs per point were conducted for the improved P-CLUSTER algorithm, but only three runs per point for the original algorithm, which consumed a much larger proportion of our allotted resources.

Fig. 4 plots the performance (core time and total time) of four different versions of P-CLUSTER when executed on the image shown in Fig. 3. Results are given for the original P-CLUSTER program as well as for P-CLUSTER with the successive incorporation of the three computational pruning techniques discussed above. For spheres of guaranteed assignment only, the maximum percentage of execution time reduction occurred when using six processors: The original version completed in a total of 3,227 seconds, whereas the modified version completed in 2,282 seconds, a 29 percent reduction. When the maximum movement effect method is included, parallelism reduces computation for up to 10–12 workstations, after which the overhead associated with communication overshadows computational gain. The percentage improvement of combining the two methods ranges from 60 percent for three processors to 33 percent for 16 processors. As shown, the maintenance of partial sums further reduced execution time by several minutes.

In order to verify that the improved clustering performance was not a consequence of one particular hardware set or operating system, we ported P-CLUSTER to an IBM SP-2 supercomputer at Argonne National Laboratory. The performance improvement trends of both the algorithmic improvements and the paralleliza-

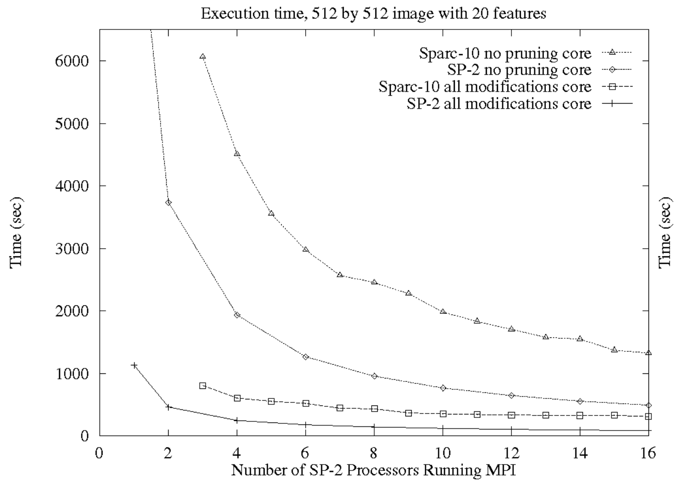


(a)

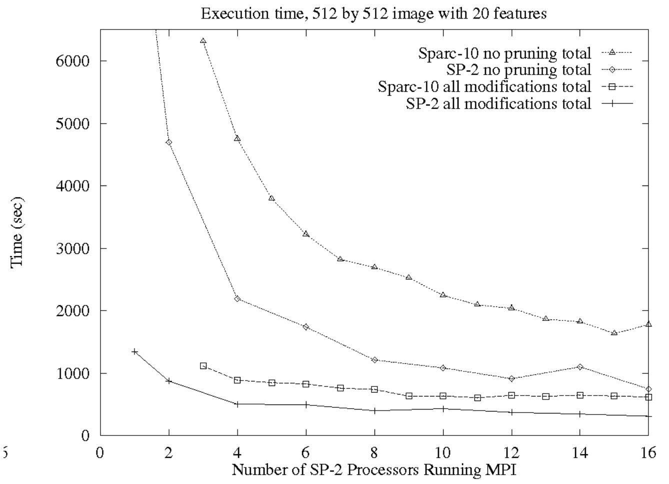


(b)

Fig. 4. Effects of three pruning techniques on 512 × 512 image, 16 clusters. (a) Core times. (b) Total times.



(a)



(b)

Fig. 5. IBM SP-2 vs. Sun Sparc-10 comparison for 512 × 512 image, 16 clusters. (a) Core times. (b) Total times.

tion were very similar to those of the trials run on the Sun Sparc-10's, although the SP-2 was considerably faster (see Fig. 5). In these experiments, MPI was used as the message passing mechanism for both the SP-2 and the Sparc-10's. The large amount of memory in each SP-2 node (128 Mbytes) also made it possible to perform clustering on the large image with only one or two processors, something that was impractical on the Sparc-10's due to thrashing. It is worth noting that the improved algorithm when executed on the slower Sparc-10 processors still completed sooner than the original algorithm when executed on much faster SP-2 processors. This observation (a better algorithm on slow systems can outperform a simple algorithm on fast systems) underscores the need for research in how to improve parallel clustering algorithms.

In order to investigate the general utility of these methods, we performed experiments on several other data sets. The data sets and the reduction in distance computations are summarized in Table 1. `Brodatz16a` [17] is the 512 × 512 image shown in Fig. 3. `Brodatz5` is an image of size 256 × 256 pixels and containing five distinct textured regions, and `Brodatz16b` is the same image as `Brodatz16a` with 45 instead of 20 features. `Coupon` and `Coupon samp` are text images passed through four filters. `Coupon samp` is a random sampling of `Coupon`; the true number of clusters is unknown. `Iris` is a well-known plant measurement data set with four clusters [2]. `Isolet` is a set of filtered speech data, specifically, individuals pronouncing the letters of the English alphabet; there are 26 expected clusters and 618 features [21].

We clustered each data set several times with different numbers of clusterings. In all cases, the proposed methods pruned away at least 69 percent of the distance computations. In many cases, the savings was much greater. For example, in the 10- and 16-cluster cases of the `Coupon` data, 96.2 percent of the distance calculations were eliminated. With the exception of the Isolet data, searching for more clusters yielded a larger computational savings. The proposed clustering algorithms also yielded larger savings with increasing problem size.

5 Conclusions

With recent advances in computer networking and data storage technologies, the demand for methods to better characterize and process large data sets is increasing. One "generic" computation that is important to many data mining applications is data clustering. Reducing clustering time enables the use of larger sampling percentages (of the population) to improve clustering accuracy and gives the researcher greater flexibility when interactively exploring data archives. Fast data clustering methods are particularly useful in image segmentation and other image processing tasks.

TABLE 1
CLUSTERING RESULTS FOR VARIOUS DATA SETS

Data Set	Domain	Number of Patterns	Number of Clusters	Number of Features	Number of Distance Calculations (thousands)		Percent Reduced
					Pre-Pruning	Post-Pruning	
Brodatz5	image	65,536	5	20	9,568	2,454	74.4
Brodatz5	image	65,536	10	20	103,600	10,930	89.5
Brodatz5	image	65,536	16	20	467,800	39,270	91.7
Brodatz16a	image	262,144	5	20	179,300	23,550	86.9
Brodatz16a	image	262,144	10	20	827,800	98,170	88.2
Brodatz16a	image	262,144	16	20	2,926,000	213,400	92.8
Brodatz16b	image	262,144	5	45	170,300	30,260	82.3
Brodatz16b	image	262,144	10	45	796,600	124,100	84.5
Brodatz16b	image	262,144	16	45	1,987,000	196,500	90.2
Coupon samp	image	4,000	5	4	1,552	181	88.3
Coupon samp	image	4,000	10	4	11,590	880	92.5
Coupon samp	image	4,000	16	4	35,590	2,632	92.7
Coupon	image	65,536	5	4	32,240	2,696	91.7
Coupon	image	65,536	10	4	583,900	22,570	96.2
Coupon	image	65,536	16	4	1,343,000	51,380	96.2
Iris	biology	150	4	4	15	4	72.9
Iris	biology	150	5	4	20	5	72.8
Iris	biology	150	16	4	177	32	82.0
Isolet	speech	6,238	5	618	6,743	1,231	81.8
Isolet	speech	6,238	16	618	43,150	10,920	74.7
Isolet	speech	6,238	26	618	75,080	22,690	69.8

We have previously developed a parallel clustering program, P-CLUSTER, designed for use on networks of workstations. In this paper, we described three computational pruning techniques that have been incorporated into P-CLUSTER. Results of experiments on two NOW platforms, a collection of Sun Sparc-10 workstations and an IBM SP-2 supercomputer, demonstrate that dramatic performance improvements in data clustering are possible by combining these pruning techniques with parallel processing.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their useful comments and suggestions on how to improve this paper. This work was supported in part by the U.S. National Science Foundation grants CDA-8806599, DUE-9551180, CCR-9503838, and CDA-9617310, and by Northrop Corporation.

A number of related papers and technical reports are available via the world wide web from two research groups at Michigan State University. Results from the Software Engineering and Network Systems Laboratory are available at http://www.cps.msu.edu/sens, and results from the Pattern Recognition and Image Processing Laboratory are available at http://www.cps.msu.edu/prip.

REFERENCES

[1] U. Fayyad, D. Haussler, and P. Stolorz, “Mining Scientific Data,” *Comm. ACM*, vol. 39, pp. 51–57, Nov. 1996.

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

[3] A.K. Jain and S. Bhattacharjee, “Text Segmentation Using Gabor Filters for Automatic Document Processing,” *Machine Vision Applications*, vol. 5, pp. 169–184, 1992.

[4] A.K. Jain and P.J. Flynn, “Image Segmentation Using Clustering,” *Advances in Image Understanding: A Festschrift for Azriel Rosenfeld*, pp. 65–83. Los Alamitos, Calif.: IEEE CS Press, 1996.

[5] H.P. Friedman and J. Rubin, “On Some Invariant Criteria for Grouping Data,” *J. Am. Statistical Assoc.*, vol. 62, pp. 1,159–1,178, 1967.

[6] D. Judd, N.K. Ratha, P.K. McKinley, J. Weng, and A.K. Jain, “Parallel Implementation of Vision Algorithms on Workstation Clusters,” *Proc. 12th Int’l Conf. Pattern Recognition*, pp. 317–321, Jerusalem, Israel, Oct. 1994.

[7] L.M. Ni and A.K. Jain, “A VLSI Systolic Architecture for Pattern Clustering,” *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 7, no. 1, pp. 80–89, Jan. 1985.

[8] K. Hwang and D. Kim, “Parallel Pattern Clustering on a multiprocessor With Orthogonally Shared Memory,” *Proc. Int’l Conf. Parallel Processing*, pp. 913–916, 1987.

[9] J.C. Tilton and J.P. Strong, “Analyzing Remotely Sensed Data on the Massively Parallel Processor,” *Proc. Seventh Int’l Conf. Pattern Recognition*, pp. 398–400, Montreal, 1984.

[10] S. Ranka and S. Sahni, “Clustering on a Hypercube Multicomputer,” *IEEE Trans. Parallel and Distributed Systems*, vol. 2, pp. 129–137, Apr. 1991.

[11] D. Judd, P.K. McKinley, and A.K. Jain, “Computational Pruning Techniques in Parallel Square-Error Clustering of Large Data Sets,” Tech. Rep. MSU-CPS-96-02, Dept. of Computer Science, Michigan State Univ., East Lansing, Mich., 1996.

[12] J.B. McQueen, “Some Methods of Classification and Analysis of Multivariate Observations,” *Proc. Fifth Berkeley Symp. Math. Statistics and Probability*, pp. 281–297, 1967.

[13] E. Forgy, “Cluster Analysis of Multivariate Data: Efficiency Versus Interpretability of Classifications,” *Biometrics*, vol. 21, p. 768, 1965.

[14] K. Fukunaga and P.M. Narendra, “A Branch and Bound Algorithm for Computing k-nearest Neighbors,” *IEEE Trans. Computers*, pp. 750–753, July 1975.

[15] H. Avi-Itzhak and T. Diep, “Lossless Acceleration for Correlation-Based Nearest-Neighbor Pattern Recognition,” *Proc. 12th Int’l. Conf. Pattern Recognition, Jerusalem*, vol. 2, pp. 240–244, 1994.

[16] V. Ramasubramanian and K. Paliwal, “Fast k-Dimensional Tree Algrotihms for Nearest Neighbor Search With Application to Vector Quantization Encoding,” *IEEE Trans. Signal Processing*, vol. 40, pp. 518–531, Mar. 1992.

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

[18] A.K. Jain and F. Farrokhnia, “Unsupervised Texture Segmentation Using Gabor Filters,” *Pattern Recognition*, vol. 24, no. 12, pp. 1,167–1,186, 1991.

[19] V.S. Sunderam, “PVM: A Framework for Parallel Distributed Computing,” *Concurrency: Practice and Experience*, vol. 2, no. 4, pp. 315–339, Dec. 1990.

[20] Message Passing Interface Forum, “MPI: A Message-Passing Interface Standard,” Tech. Rep. CS-94-230, Dept. of Computer Science, Univ. of Tennessee, Knoxville, Tenn., May 1994.

[21] M. Fanty and R. Cole, “Spoken Letter Recognition,” *Advances in Neural Information Processing Systems 3*, San Mateo, Calif., 1991.